00 - Introduction

François Pitié

Ussher Assistant Professor in Media Signal Processing
Department of Electronic & Electrical Engineering, Trinity College Dublin

[4C16] Machine Learning with Applications in Media Engineering

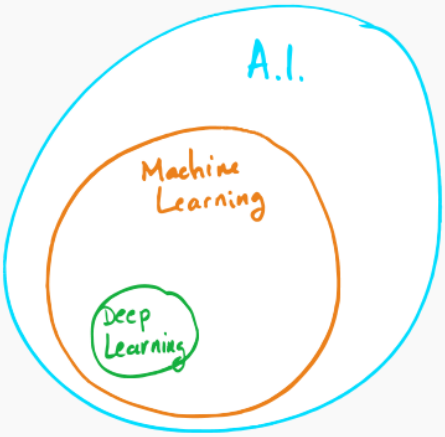
This module is an introduction to Machine Learning and especially Deep Neural Nets, which are disrupting all aspects of society today.

The material is constructed in collaboration with leading industrial practitioners including Google, YouTube and Movidius/Intel.

Hands on labs will give you experience with these applications.

Deep Learning is a particular type of **machine learning** method, and is thus part of the broader field of **artificial intelligence** (using computers to reason).

Deep learning is another name for **artificial neural networks**, which are a loosely inspired by the structure of the neurons in the cerebral cortex.



The recent quantum leap in machine learning has solely been driven by deep learning successes.

When you read or hear about AI or machine Learning successes in recent years, it really means Deep Learning successes.

Machine Learning can be split into 3 main fields:

Supervised Learning:

From a labelled dataset $(\mathbf{x}_i, y_i)$, we want to infer $f(\mathbf{x}_i) = y_i$

Unsupervised Learning:

What can we learn about a dataset $(\mathbf{x}_i)$ by just looking at it? (ie. without any labelled information y_i)

Reinforcement Learning:

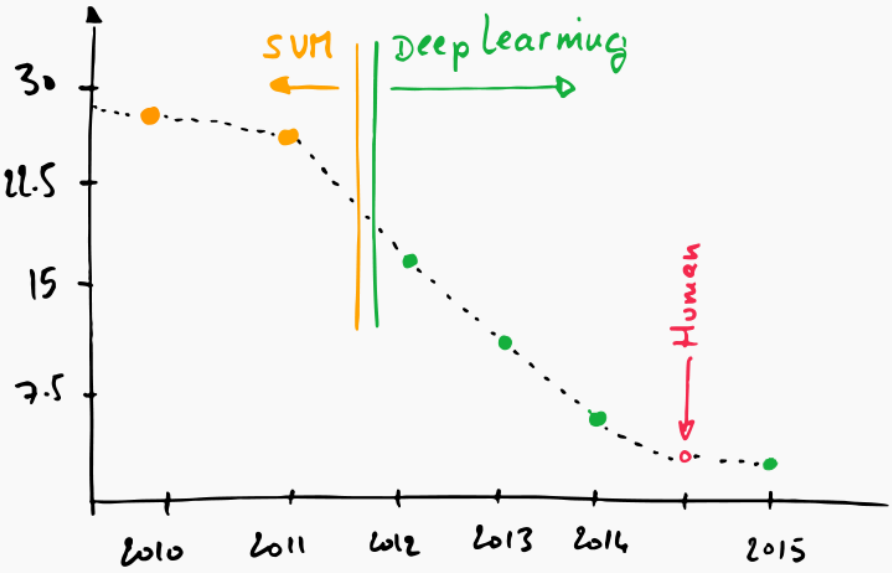
How can an agent interact with its environment (the data) to get maximum reward (eg. game playing, robots learning to walk).

Deep Learning has made major breakthroughs in all three fields.

Deep Learning Successes

Image Recognition is one of the core applications of Computer Vision. ImageNet [www.image-net.org] runs an annual challenge where software programs compete to correctly classify and detect objects and scenes in images.

The error rate in object recognition for that challenge has massively dropped since the introduction of deep neural networks in 2012 [1]. Machines can now do better than humans.

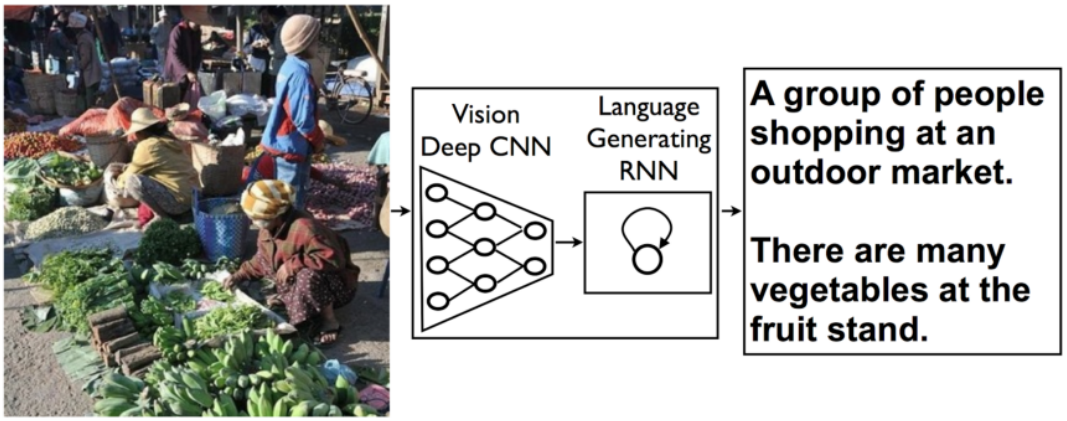


[1] ImageNet Classification with Deep Convolutional Neural Networks
A Krizhevsky, I Sutskever, G Hinton, 2012 [https://goo.gl/wxen2Y]

Neural nets are also advancing the state of the art in Scene Understanding.

[1] Conditional Random Fields as Recurrent Neural Networks
S Zheng et al., 2015 [https://arxiv.org/abs/1502.03240]

Image models combined with language models make it possible to automatically generate captions from images.



Google Research Blog [https://goo.gl/U88bDQ]

All major tech companies have changed their machine translation systems to use Deep Learning.

Google used to average a yearly 0.4% improvement on their machine translation system. Their first attempt at using Deep Learning yielded an overnight 7% improvement! More than in an entire lifetime!

Several years of handcrafted development could not match a single initial deep learning implementation.

> New York Times, “The Great AI Awakening” [https://goo.gl/DPYp6d]

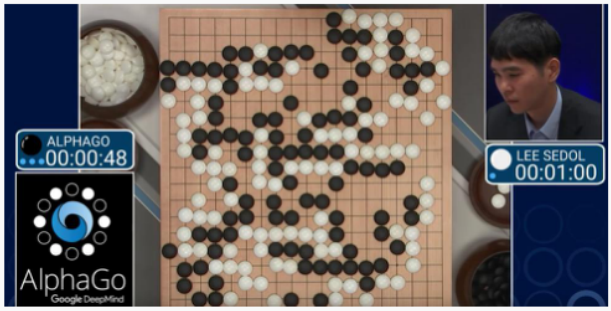
In 2014, Skype Translator was announced. It trains and optimises speech recognition, automatic machine translation and speech synthesis tasks, acting as the glue that holds these elements together.

> Skype demo [https://www.youtube.com/embed/NhxCg2PA3ZI?start=0]
> Microsoft blog post [https://goo.gl/eAuPcS]

Deep learning has also been introduced in reinforcement learning to solve complex sequential decision making problems.

Recent successes include:

playing old Atari computer games, programming real world Robots and beating humans at Go.



> demo: Robots Learning how to walk https://www.youtube.com/hx_bgoTF7bs
>
> DeepMind [https://goo.gl/3TcCNA]

Reasons of a Success

Neural Networks have been around for decades. But is only now that it surpasses all other machine learning techniques.

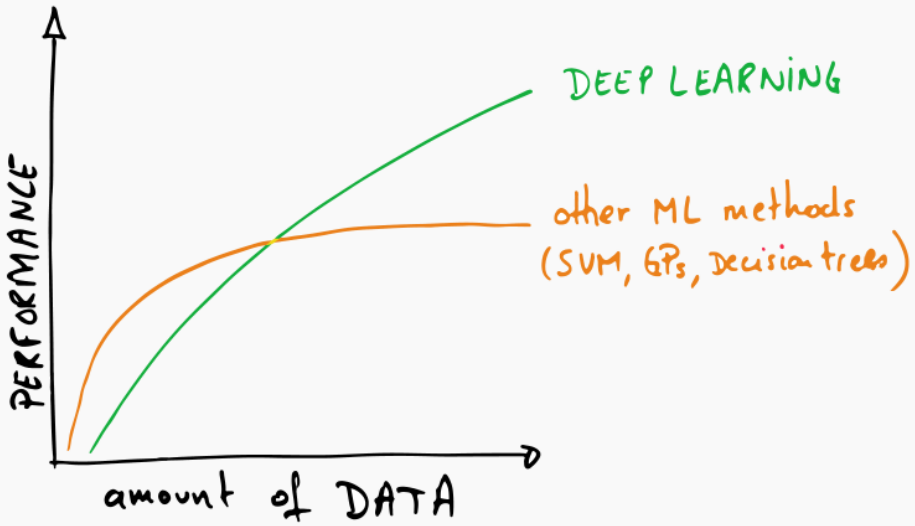
Deep Learning is now a **disruptive** technology that has been unexpectedly taking over operations of technology companies around the world.

"The revolution in deep nets has been very profound, it definitely surprised me, even though I was sitting right there."

— Sergey Brin, Google co-founder

Why now?

Because Deep Learning does scale.



Neural Nets are the only ML technique whose performance scales efficiently with the training data size. Other ML popular techniques just can't scale that well.

The advent of **big databases**, combined with cheaper **computing power** (Graphic Cards), meant that Deep Learning could take advantage of all this, whilst other techniques stagnated. Instead of using thousands of observations, Deep Learning can take advantage of billions.

The tipping point was 2012 in Computer Vision and around 2014 in Machine Translation.

Deep Learning offers a (relatively) simple framework to define and parametrise pretty much any kind of numerical method and then optimise it over massive databases.

By adopting an automated brute force approach to tuning algorithms, Deep Learning is able to surpass hand-tailored algorithms of skilled researchers.

It offers a systematic approach when before algorithms took years of human efforts to design.

Reasons of a Success

Democratisation

Deep Learning is a (relatively) simple framework.

Good programmers train state of the art neural nets without having done 10+ years of research in the domain.

It is an opportunity for start-ups and it has become a ubiquitous tool in tech companies.

Reasons of a Success

Global Reach

It has been applied successfully to many fields of research, industry and society:

self-driving cars, image recognition, detecting cancer, speech recognition, speech synthesis, machine translation, drug discovery and toxicology, customer relationship management, recommendation systems, bioinformatics, advertising, controlling lasers, etc.

Impact

Here is a question for you:

How long before your future job gets replaced by an algorithm?

Probably much sooner than you think. You might feel safe if you are an artist...

… but then again:

automatic style transfer [1]

[1] A Neural Algorithm of Artistic Style
L. Gatys, A. Ecker, M. Bethge, 2015 [https://arxiv.org/abs/1508.06576]

Does an AI need to make love to Rembrandt's girlfriend to make art? [https://goo.gl/gi7rWE]

Intelligent Machines: AI art is taking on the experts [https://goo.gl/2kfyXd]

4C16: Course Structure

Course Content

Part 1. Machine Learning Fundamentals

In Week 1 - 4, we will cover

Least Squares: the root of all Machine Learning.
(feature mapping, over/under fitting, regularisation, maximum likelihood)

Logistic Regression: your first Neuron.
(linear classifier, cross-entropy , gradient descent optimisation)

Classic Classifiers: overview of classic machine learning algorithms.
(SVM, Decision Trees, Kernel Trick, Nearest-Neighbours)

Comparing Classifiers:
(ROC curves, confusion tables, F1 score)

Course Content

Part 2. Neural Net Fundamentals

Feedforward Neural Network
(network architecture, back-propagation, regularisation, vanishing gradients)

Convolutional Neural Network
(convolution layers, pooling, visualisation, knowledge transfer)

Autoencoders
(unsupervised learning, PCA, CNN autoencoder)

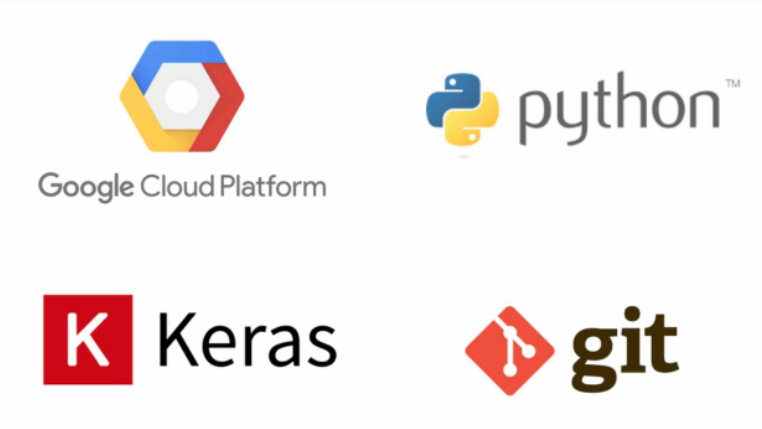
Recurrent Neural Network
(LSTM, encodings, text processing)

Assignments

Week 2	Least Squares lab	[2% mark]
Week 3	Logistic Regression lab	[2% mark]
Week 4	Classifiers lab	[2% mark]
Week 5	Feed Forward Neural Net lab	[2% mark]
Week 6	Convolutional Neural Net lab	[2% mark]
Week 8	**Quiz**	**[5% mark]**
Week 8-10	lab project	[5% mark]
Week 9-11	lab project	[5% mark]
End of term	**Exam**	**[75% mark]**

Labs

We have developed a fantastic environment specially for you, so that you can learn best industry practices.



Labs

You will be programming in python 3 using Keras and TensorFlow. Everything will be running on the Google Cloud Platform, which gives you on-demand scalable computing resources.

Your coding environment will be a combination of shell/terminal, editor, and Jupyter notebook.

You will use Git to checkpoint your progress and for continuous feedback on lab assignments.

Books & Resources

[1] Deep Learning (MIT press)
Ian Goodfellow et al.
[https://www.deeplearningbook.org]

[2] Machine Learning on Cousera
Andrew Ng
[https://www.coursera.org/learn/machine-learning]

[3] Neural Networks and Deep Learning
Michael Nielsen
[http://neuralnetworksanddeeplearning.com/]

[4] Curated list of links https://github.com/ChristosChristofidis/awesome-deep-learning

[5] Brandon Rohrer's YT channel https://www.youtube.com/user/BrandonRohrer

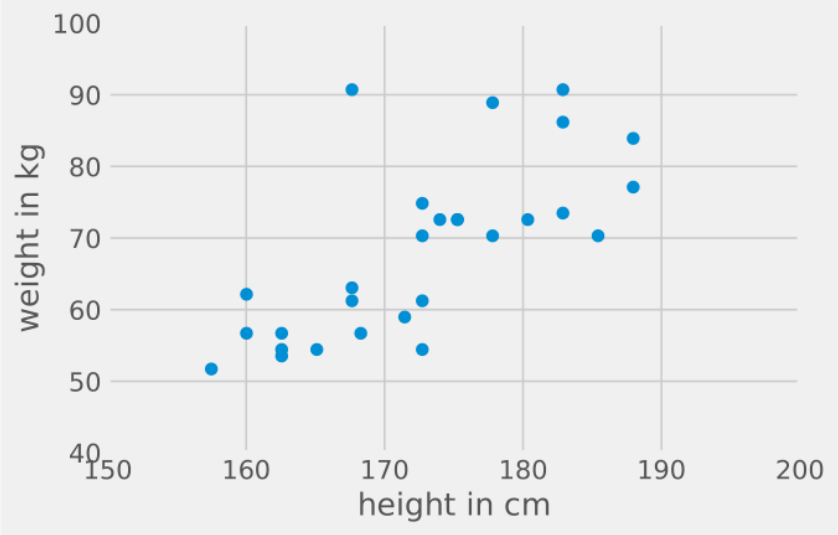
[6] Siraj Raval's YT channel https://www.youtube.com/channel/UCWN3xxRkmTPmbKwht9FuE5A

01 - Linear Regression/Least Squares

François Pitié

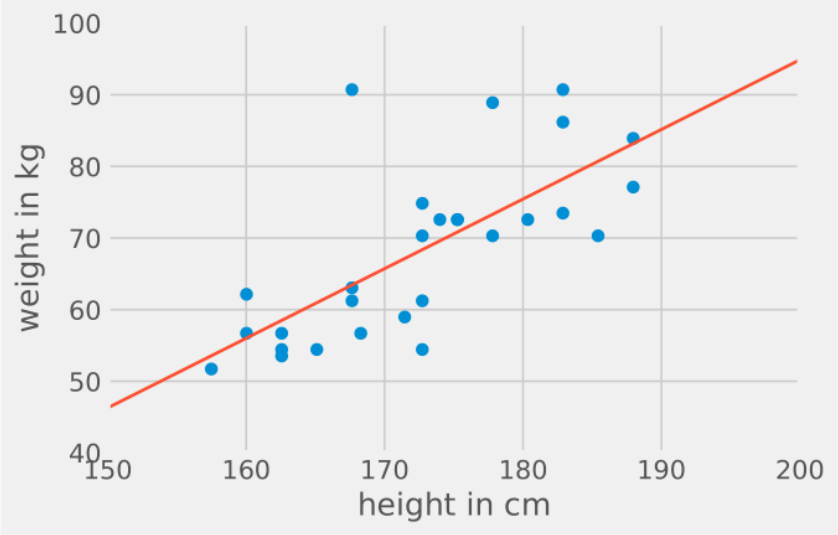
Ussher Assistant Professor in Media Signal Processing
Department of Electronic & Electrical Engineering, Trinity College Dublin

Linear Regression



We have collected some data.

Linear Regression



We are looking to infer a linear prediction:

$$\text{weight(kg)} = \text{height(cm)} \times 0.972 - 99.5$$

Linear Regression $\Longleftrightarrow$ Least Squares

Linear Regression

The **input** is a feature vector $(x_1, \cdots, x_p)$.

The **output** is a scalar y. (It is easy to generalise to a vector output by splitting the vector output into multiple scalar outputs.)

Our **model** links the output y to the input feature vector $(x_1, \cdots, x_p)$ with a linear relationship:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_p x_p$$

$\hat{y}_i = w_i x_i = \text{prediction}$

Linear Regression

In practice, you have n observations, for which you have extracted p features:

$$
\begin{aligned}
y_1 &= w_0 + w_1 x_{11} + w_2 x_{12} + w_3 x_{13} + \cdots + w_p x_{1p} + \varepsilon_1 \\
y_2 &= w_0 + w_1 x_{21} + w_2 x_{22} + w_3 x_{23} + \cdots + w_p x_{1p} + \varepsilon_2 \\
y_3 &= w_0 + w_1 x_{31} + w_2 x_{32} + w_3 x_{33} + \cdots + w_p x_{3p} + \varepsilon_3 \\
&\vdots \\
y_n &= w_0 + w_1 x_{n1} + w_2 x_{n2} + w_3 x_{n3} + \cdots + w_p x_{np} + \varepsilon_n
\end{aligned}
$$

As the model can't explain everything we introduce an error term ε.

Linear Regression

We want to find $w_0, w_1, \cdots, w_p$ that minimises the error.

At this point the error $(\varepsilon_i)_{1 \le i \le n}$ is a vector of n separate terms. Since we can't minimise a vector, we need to aggregate the values into a single scalar that be used for comparison.

In linear regression, we choose to combine the error terms using the mean squared error (MSE):

Loss Function

$$E = \frac{1}{n}\sum_{i=1}^{n} \varepsilon_i^2 = \frac{1}{n}\sum_{i=1}^{n} \left(w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} - y_n\right)^2$$

The choice of the mean squared error is a fundamental aspect of linear regression. Other error metrics are possible (eg. mean absolute difference) but the they lead to very different mathematics.

Least Squares

⇒ Supervised Learning

x is the feature vector
$x_1, x_2, x_3 \dots$ are features

$x = (x_1, x_2, x_3, x_4 \dots)$

x_1: weight (continuous)
x_2: gender (categorical)

$f(x_1, \dots, x_p) \rightarrow y$ (scalar)

— not 'n', 'p'!

$w_1, w_2 \dots$ are model parameters

$$f(x_1, \dots, x_p) = w_1 x_1 + \dots + w_p x_p + w_0$$

w_0 known as the bias

$$y_1 = w_0 + w_1 x_{11} + w_2 x_{12} + \dots + w_p x_{1p} + \varepsilon_1$$
$$y_2 = w_0 + w_1 x_{21} + w_2 x_{22} + \dots + w_p x_{2p} + \varepsilon_2$$
$$\vdots$$
$$y_n = w_0 + w_1 x_{n1} + w_2 x_{n2} + \dots + w_p x_{np} + \varepsilon_n$$

Error

- 'n' is the number of 'observations'
- 'p' is the no. of 'features'
- Error terms are a vector which cannot be compared against one another
  ⇒ Mean Square Error

These calculations are DONE in the notes

Loss Function (MSE)

$$E(w_0, \dots, w_p) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i^2 - y_i^2)$$
$$= \frac{1}{n} \sum_{i=1}^{n} (w_0 + w_1 x_{i1} + w_2 x_{i2} + \dots + w_p x_{ip} - y_i)^2$$

$\frac{\partial E}{\partial w_0} = 0, \dots, \frac{\partial E}{\partial w_p} = 0$ ← minimize error across all features

$$\frac{\partial E}{\partial w_k} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial}{\partial w_k} (w_0 + w_1 x_{i1} + \dots + w_p x_{ip} - y_i)^2 = 0$$

$$w_0 : \frac{2}{n} \sum_{i=1}^{n} x_{i0} (w_0 + \dots + w_p x_{ip} - y_i) = 0$$

w_1 :

$\vdots$

$$w_p : \frac{2}{n} \sum_{i=1}^{n} x_{ip} (w_0 + \dots + w_p x_{ip} - y_i) = 0$$

} p+1 equations

$$w_0\left(\frac{2}{n}\sum_{i=1}^{n} x_{i0}\right) + w_1\left(\frac{2}{n}\sum_{i=1}^{n} x_{i0}x_{i1}\right) + \cdots + w_p\left(\frac{2}{n}\sum_{i=1}^{n} x_{i0}x_{ip}\right) = y_{ix}$$

Derivation Calculus

Now let us derive the optimal values for $w_0, w_1, \cdots, w_p$ that minimise the mean squared error function $E(w_0, w_1, \cdots, w_p)$.

error is a function of the weights → features are constant for each observation

minimizing the weights

Derivation Calculus

$$E(w_0, \cdots, w_p) = \frac{1}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} - y_n\right)^2$$

At the minimum of E, $\frac{\partial E}{\partial w_0} = \cdots = \frac{\partial E}{\partial w_p} = 0$:

Simple chain rule

$$\frac{\partial E}{\partial w_0}(w_0, \cdots, w_p) = \frac{2}{n}\sum_{i=1}^{n}\left(w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} - y_i\right) = 0$$

x_{io} is assumed to be 1

$$\frac{\partial E}{\partial w_1}(w_0, \cdots, w_p) = \frac{2}{n}\sum_{i=1}^{n} x_{i1}\left(w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} - y_i\right) = 0$$

$$\vdots$$

$$\frac{\partial E}{\partial w_p}(w_0, \cdots, w_p) = \frac{2}{n}\sum_{i=1}^{n} x_{ip}\left(w_0 + w_1 x_{i1} + \cdots + w_p x_{ip} - y_i\right) = 0$$

Derivation Calculus

Rearranging terms and dividing by $2/n$:

$$
\begin{aligned}
w_0 \sum_{i=1}^{n} 1 \quad &+ w_1 \sum_{i=1}^{n} x_{i1} \quad + \cdots + w_p \sum_{i=1}^{n} x_{ip} \quad = \sum_{i=1}^{n} y_i \\
w_0 \sum_{i=1}^{n} x_{i1} \; &+ w_1 \sum_{i=1}^{n} x_{i1}^2 \quad + \cdots + w_p \sum_{i=1}^{n} x_{i1} x_{ip} = \sum_{i=1}^{n} x_{i1} y_i \\
&\vdots \qquad\qquad \vdots \quad \vdots \qquad\qquad \vdots \\
w_0 \sum_{i=1}^{n} x_{ip} \; &+ w_1 \sum_{i=1}^{n} x_{ip} x_{i1} + \cdots + w_p \sum_{i=1}^{n} x_{ip}^2 \quad = \sum_{i=1}^{n} x_{ip} y_i
\end{aligned}
$$

This gives us a linear system of $p + 1$ equations, which can be solved efficiently using linear solvers.

Matrix Calculus

We are now going to derive the same equations using matrix notations. It is useful in practice to know how to do this without having to come back to these sums and system of equations.

Notations

By convention, we write a scalar as x, a vector as $\mathbf{x}$ and a matrix as $\mathbf{X}$.
We denote: (or $\vec{x}$)

Outcome Vector

Feature Vectors

$$\mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad \mathbf{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix}, \quad \mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_p \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

Weight Vector (model parameters)

Error Vector

The linear model then becomes:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$$

The matrix $\mathbf{X}$, which stacks all the observations, is also called the Design Matrix.

p features
n observations

$$y_i = w_0 + w_1 x_{i1} + \dots + w_p x_{ip}$$

$$\Rightarrow y_i = x_i^T w$$

$$\begin{cases} y_1 = x_1^T w \\ \vdots \\ y_n = x_n^T w \end{cases} \iff \hat{y} = X w \quad \text{where} \quad y = Xw + \varepsilon$$

Matrix Notations

In matrix notations, the mean squared error can be written as:

$$E(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n}\varepsilon_i^2 = \frac{1}{n}\varepsilon^\top\varepsilon = \frac{1}{n}\|\varepsilon\|^2$$

$$= \frac{1}{n}(\mathbf{X}\mathbf{w}-\mathbf{y})^\top(\mathbf{X}\mathbf{w}-\mathbf{y})$$

$$= \frac{1}{n}\left(\mathbf{w}^\top\mathbf{X}^\top\mathbf{X}\mathbf{w} + \mathbf{y}^\top\mathbf{y} - 2\mathbf{w}^\top\mathbf{X}^\top\mathbf{y}\right)$$

At the minimum of $E(\mathbf{w})$, we have

$$\frac{\partial E}{\partial \mathbf{w}} = \left(\frac{\partial E}{\partial w_0}, \cdots, \frac{\partial E}{\partial w_p}\right) = (0, \cdots, 0)$$

$\frac{\partial E}{\partial \mathbf{w}}$ is the gradient of E and is often denoted as ∇E

Gradient Calculus

Knowing how to derive the gradient in matrix notations is important. Below is a list of useful gradient derivations.

We assume $\mathbf{a}, \mathbf{b}, \mathbf{A}$ are independent of $\mathbf{w}$.

(matrix; vectors)

(1) * $\dfrac{\partial \mathbf{a}^\top \mathbf{w}}{\partial \mathbf{w}} = \mathbf{a}$

(2) * $\dfrac{\partial \mathbf{b}^\top \mathbf{A}\mathbf{w}}{\partial \mathbf{w}} = \mathbf{A}^\top \mathbf{b}$

(3) * $\dfrac{\partial \mathbf{w}^\top \mathbf{A}\mathbf{w}}{\partial \mathbf{w}} = (\mathbf{A} + \mathbf{A}^\top)\mathbf{w}$ (or $2\mathbf{A}\mathbf{w}$ if A symmetric)

(4) * $\dfrac{\partial \mathbf{w}^\top \mathbf{w}}{\partial \mathbf{w}} = 2\mathbf{w}$

(5) $\dfrac{\partial\, \mathbf{a}^\top \mathbf{w}\mathbf{w}^\top \mathbf{b}}{\partial\, \mathbf{w}} = (\mathbf{a}\mathbf{b}^\top + \mathbf{b}\mathbf{a}^\top)\mathbf{w}$

$$a^\top w = \sum_{k=0}^{n} a_k w_k \rightarrow \frac{\partial a^\top w}{\partial w_k} = \frac{\partial}{\partial w_k} \sum_{k=0}^{n} a_k w_k$$

$$= a_k$$

$$\Rightarrow \frac{\partial a^\top w}{\partial w} = \begin{pmatrix} a_1 \\ \vdots \\ a_p \end{pmatrix} = \vec{a}$$

Exercise:

compute the gradient $\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$ for

$$E(\mathbf{w}) = (\mathbf{w} - \mathbf{B}\mathbf{w})^\top \mathbf{A}(\mathbf{w} - \mathbf{a})$$

We have no assumptions about matrices $\mathbf{A}$ and $\mathbf{B}$.

$$\frac{\partial E(w)}{\partial w} = \frac{\partial}{\partial w}(w - Bw)^T A(w - a)$$

$$= \frac{\partial}{\partial w}(w^T - w^T B^T)(Aw - Aa)$$

$$= \frac{\partial}{\partial w}(w^T Aw) - \frac{\partial}{\partial w}(w^T Aa) - \frac{\partial}{\partial w}(w^T B^T Aw) + \frac{\partial}{\partial w}(w^T B^T Aa)$$

Matrix Derivations

Let's come back to our problem:

$$\frac{\partial E}{\partial \mathbf{w}} = \frac{1}{n}\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} + \mathbf{y}^\top \mathbf{y} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y}\right)$$

Apply the previous slide formula for each of the terms:

$$\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w}\right) = 2\mathbf{X}^\top \mathbf{X}\mathbf{w} \quad (3) \rightarrow X^\top X \Leftrightarrow A$$

$$\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{y}^\top \mathbf{y}\right) = 0 \quad \text{no dependence on } w$$

$$\frac{\partial}{\partial \mathbf{w}}\left(\mathbf{w}^\top \mathbf{X}^\top \mathbf{y}\right) = \mathbf{X}^\top \mathbf{y} \quad (2) \rightarrow w^\top X^\top y \xrightarrow{I} = y^\top X w$$

Thus

$$\frac{\partial E}{\partial \mathbf{w}} = \frac{2}{n}\mathbf{X}^\top \mathbf{X}\mathbf{w} - \frac{2}{n}\mathbf{X}^\top \mathbf{y} = 0$$

which can be simplified as follows:

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

Matrix Derivations

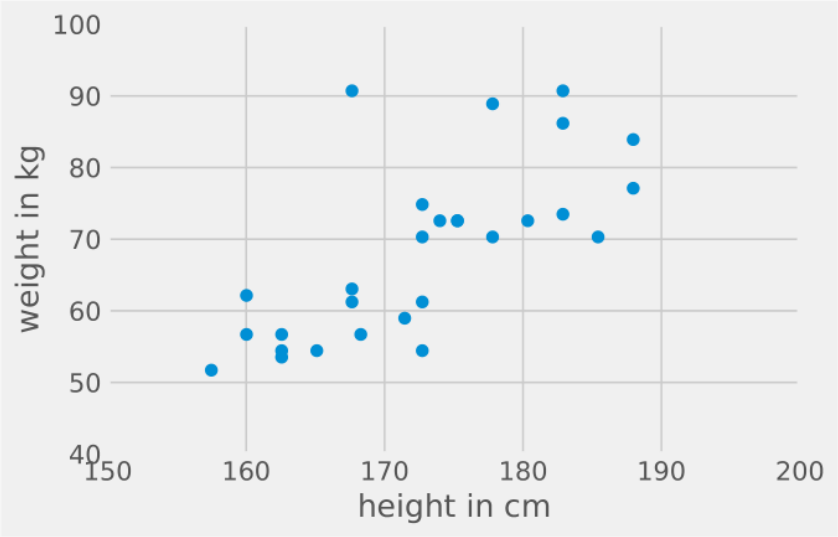
This is called the normal equation:

$$\mathbf{X}^\top \mathbf{X}\mathbf{w} = \mathbf{X}^\top \mathbf{y}$$

and it is the same as our linear system in slide 9.

Curve Fitting

Let’s come back to our original …



Curve Fitting

...and derive the normal equations using matrix notations.

The model is affine $\underbrace{y = w_0 + w_1 x}_{\text{model}}$.

The design matrix that stacks all features is thus $\mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}$

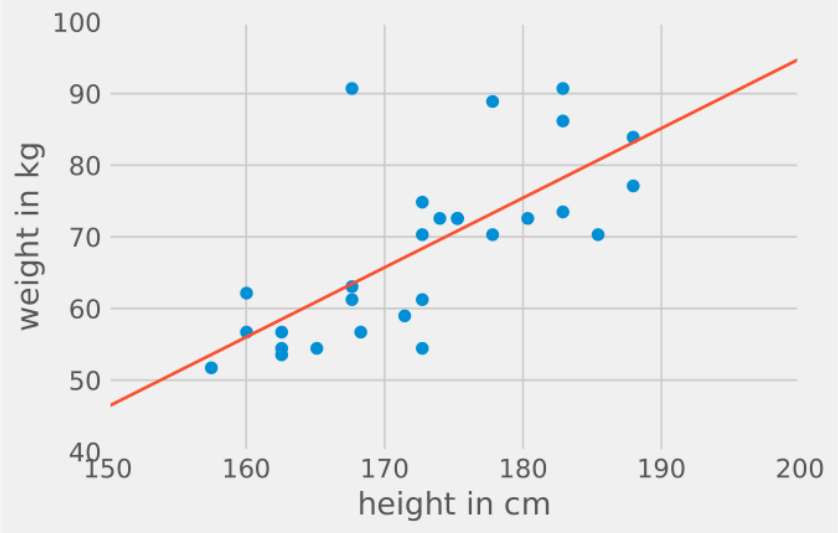
and the matrices of the normal equations are:

$$\mathbf{X}^\top \mathbf{X} = \begin{pmatrix} \sum_{i=1}^n 1 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{pmatrix}, \quad \mathbf{X}^\top \mathbf{y} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{pmatrix}$$

The LS estimate is then:

$$\hat{\mathbf{w}} = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} = \begin{pmatrix} \sum_{i=1}^n 1 & \sum_{i=1}^n x_i \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \end{pmatrix}$$

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



We find $\hat{\mathbf{w}} = \begin{pmatrix} -99.5 \\ 0.972 \end{pmatrix}$. Thus our linear model is:

$$\text{weight} = \text{height} \times 0.972 - 99.5$$

Curve Fitting

Although the model is linear, it doesn't mean that we can only fit a linear or affine curve.

Consider the following polynomial model:

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

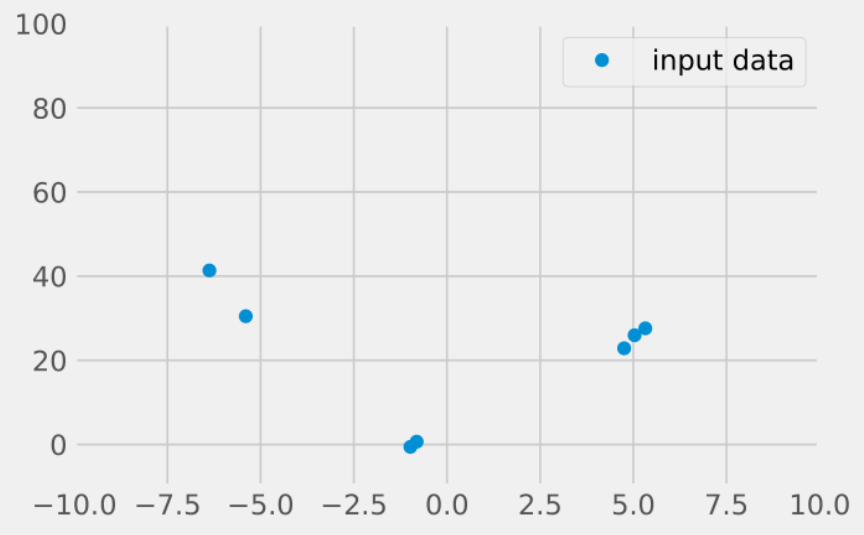
This is still a "linear" model in the sense that y is still a linear combination of 1, x, x^2 and x^3.

Many other basis decomposition can be used, e.g.

$$y = w_0 + w_1 \cos(2\pi x) + w_2 \sin(2\pi x)$$

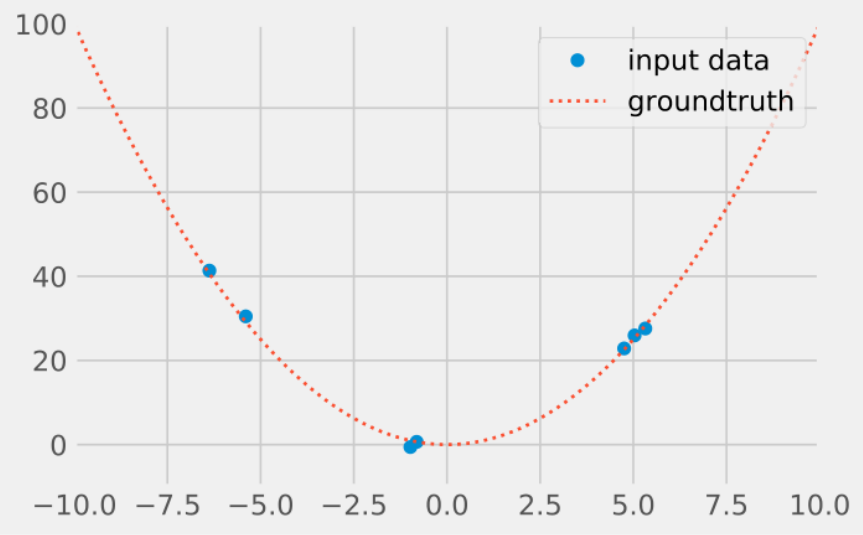
Curve Fitting

Here is an example of using Least Square for polynomial fitting.



Curve Fitting

The true model is of the form: $y = w_0 + w_1 x + w_2 x^2$



Curve Fitting

Let's derive the normal equations using matrix notations.

The model is $y = w_0 + w_1 x + w_2 x^2$, thus the features are

$$\mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{pmatrix}$$

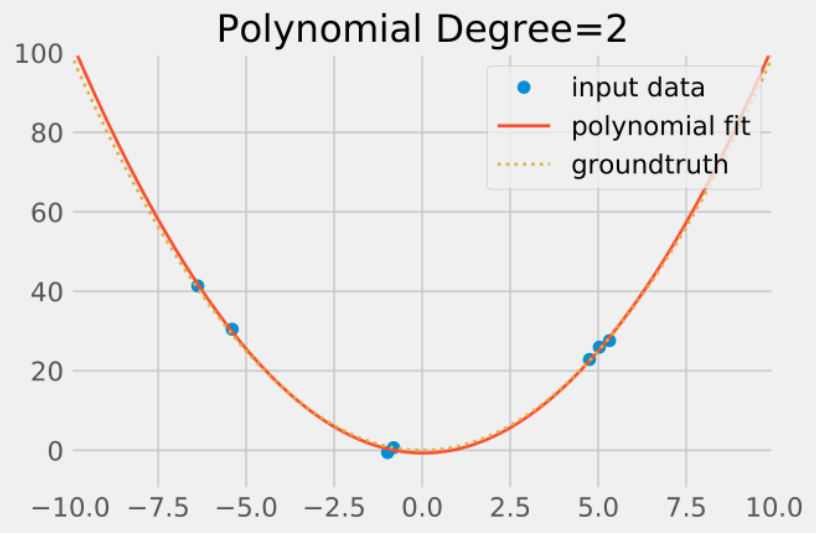
$$\mathbf{X}^\top \mathbf{X} = \begin{pmatrix} \sum_{i=1}^n 1 & \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 \\ \sum_{i=1}^n x_i & \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 \\ \sum_{i=1}^n x_i^2 & \sum_{i=1}^n x_i^3 & \sum_{i=1}^n x_i^4 \end{pmatrix}, \quad \mathbf{X}^\top \mathbf{y} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n x_i y_i \\ \sum_{i=1}^n x_i^2 y_i \end{pmatrix}$$

The LS estimate is then:

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

Curve Fitting

This is what the LS estimate looks like:

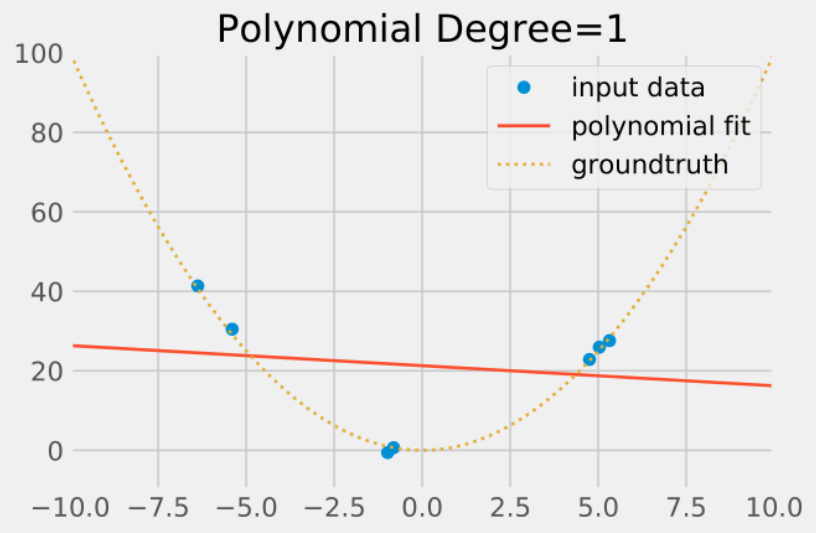


MSE: 4.38e-01

Loss Function ϵ

underfitting

Let's see what happens when you try to fit the data with a lower model order: $y = w_0 + w_1 x$



MSE: 2.02e+02

underfitting

This problem is called underfitting. This is a frequent problem in machine learning.

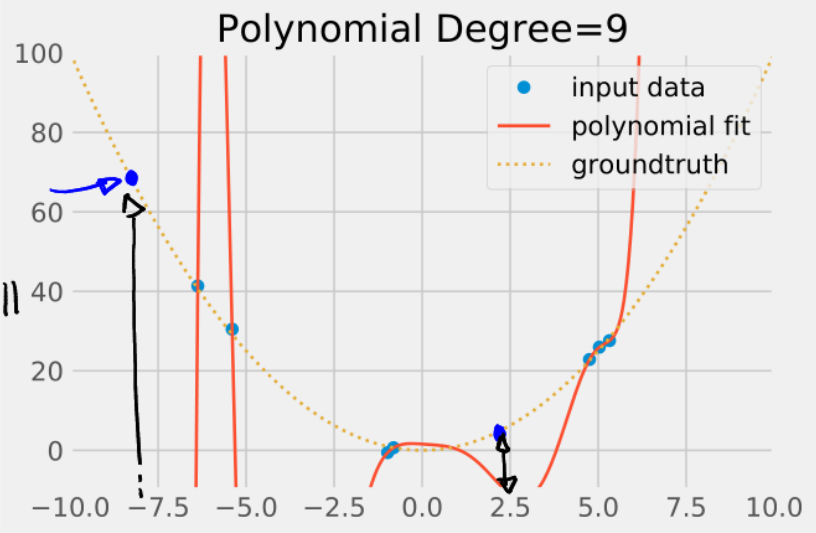
How do you know that your are underfitting?

You know that you are underfitting when the error cannot get low enough.

overfitting

Let's now try a higher model order: $y = w_0 + w_1 x + \cdots + w_9 x^9$

Adding data outside the training set will expose the overfitting



MSE: 7.59e-06

overfitting

Although the error on the observed data is perfect (MSE=7.59e-06), it is clear that the predicted model is grossly wrong in-between the observed data.

This problem is called overfitting and is a fundamental problem in machine learning.

It boils down to this: given enough parameters your model will fit pretty much anything. But that doesn't mean your model can generalise well to any data outside the data used for training.

How to know if you are overfitting?

You know that you are overfitting when the error is very low on the data used for training but quite high on newly predicted data.

How to combat overfitting?

The first thing to consider is to check if the chosen model is too complex for the data. Thus use a simpler model and make sure you are not under-fitting.

eg: you are fitting a polynomial of order 9 but the model is in fact of order 2

How to combat overfitting?

Sometimes the model is correct but you simply don't have enough observations to fit our model.

The cure is then to get more data.

eg. you only use 5 points to fit a polynomial of order 9, you need more data.

How to combat overfitting?

Using plenty of data even allows you to use overly complex models. If some features are not useful, you can expect that the corresponding estimated weights w_i will shrink to zero.

Thus it is OK to fit a polynomial of order 9 when the underlying model is actually of order 2. Just make sure you have plenty of data.

How to combat overfitting?

But what if you can’t get enough data?

Get more.

How to combat overfitting?

But what if really can't?

One last catch-all solution is to use regularisation.

Tikhonov Regularisation

In Least Squares, a natural regularisation technique is called the Tikhonov regularisation.

Instead of minimising $\|\varepsilon\|^2 = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$, we minimise a slightly modified expression:

$(\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y})$

$$E'(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \alpha\|\mathbf{w}\|^2$$

"Regularisation Term" $= \mathbf{w}^T\mathbf{w}$

This extra term improves the conditioning of the problem (ie. making the matrix $\mathbf{X}^T\mathbf{X}$ invertible) whilst still having a direct solution:

$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X} + \alpha\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}$$

Singular = hard to invert

$\underbrace{\mathbf{X}^T\mathbf{X}}_{\text{close to singular}}\mathbf{w} = \mathbf{X}^T\mathbf{y}$ (normal eqn)

where $\mathbf{I}$ is identity matrix (zeros everywhere and ones on the diagonal).

Regularisation

Basically the effect of the Tikhonov regularization is to penalise the parameters $\mathbf{w}$ when it is far away from 0. It is a bias that pulls the estimation of $\mathbf{w}$ slightly towards $\mathbf{0}$.

The motivation is that, given no other information, it is more likely that the weights $\mathbf{w}$ are small than high.

eg. it is more likely to have

$$\text{weight} = \text{height} \times \overset{w}{0.972} - 99.5$$

than

$$\text{weight} = \text{height} \times \overset{w}{10^{10}} - 10^{20}$$

Which would be preferrable?
↳ smaller w

Thus, as a default, we should favour weights $\mathbf{w}$ that are closer to zero.

$$\frac{\partial \mathcal{E}'(w)}{\partial w} = \vec{0} = \frac{\partial \mathcal{E}(w)}{\partial w} + \frac{\partial}{\partial w} \alpha w^T w$$

$$= 2X^T(Xw - y) + 2\alpha w$$

$$= 2X^T X w - 2X^T y + 2\alpha w$$

$$= 2w(X^T X + \alpha I) - 2X^T y$$

I = identity matrix

$$(X^T X + \alpha I)w = X^T y$$

doesn't change w

Regularisation

Regularisation is often a necessary evil. It allows you to avoid gross errors when predicting samples that are far outside the range of the training data.

But this comes at the cost of biasing the estimation.

Thus in practice you want to avoid it.

A good way to avoid regularisation is to get enough data so that $\mathbf{X}^\mathsf{T}\mathbf{X}$ becomes comfortably invertible.

So get plenty data!

Maximum Likelihood

Very early on, Gauss connected Least squares with the principles of probability and to the Gaussian distribution.

Maximum Likelihood

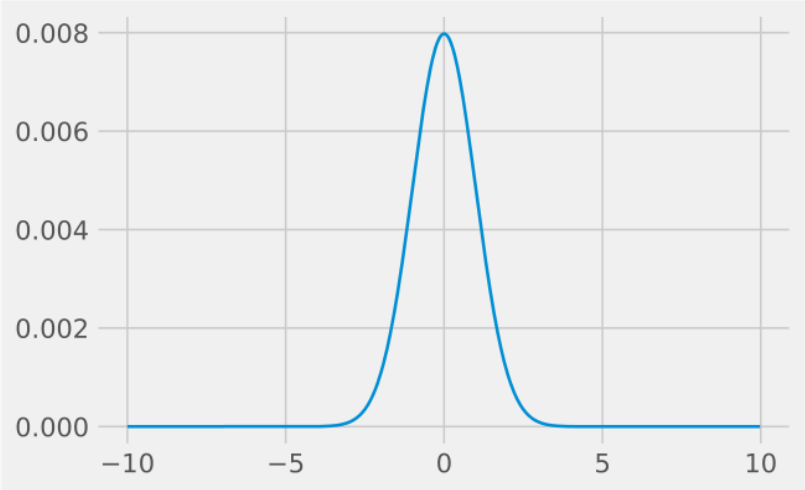
Recall that the linear model is:

$$\mathbf{y} = \mathbf{X}\mathbf{w} + \varepsilon$$

Let's give a probabilistic view on this by assuming that the error ε follows a Gaussian distribution:

$$\varepsilon \sim \mathcal{N}(0, \sigma^2)$$

$$p(\varepsilon) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{\varepsilon^2}{2\sigma^2}}$$



Maximum Likelihood

The likelihood to have y_i given $\mathbf{x}_i$ is

error being = prediction − real

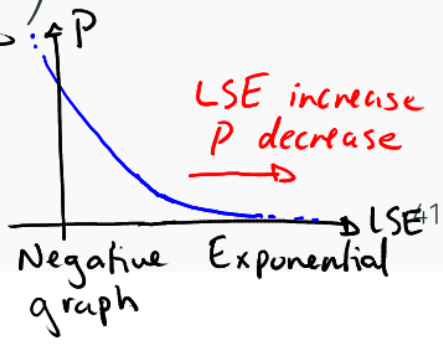
$$p(y_i|\mathbf{x}_i, \mathbf{w}) = p(\varepsilon_i = \mathbf{x}_i^\top \mathbf{w} - y_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{(\mathbf{x}_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}\right)$$

The likelihood to have all outputs $\mathbf{y}$ given all data $\mathbf{X}$ is given by

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}) = \prod_{i=1}^{n} p(\varepsilon_i)$$

$$= \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^n \exp\left(-\sum_{i=1}^{n} \frac{\left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2}{2\sigma^2}\right)$$

LS Loss Function



Maximum Likelihood

We seek to find the maximum likelihood estimate of $\mathbf{w}$. That is, finding $\mathbf{w}$ that maximises the likelihood $p(\mathbf{y}|\mathbf{X}, \mathbf{w})$:

$$\hat{\mathbf{w}}_{ML} = \arg\max_{\mathbf{w}} p(\mathbf{y}|\mathbf{X}, \mathbf{w})$$

A more practical, but equivalent, approach is to minimise the negative log likelihood:

$$\begin{aligned}
\hat{\mathbf{w}}_{ML} &= \arg\min_{\mathbf{w}} -\log\left(p(\mathbf{y}|\mathbf{X}, \mathbf{w})\right) \\
&= \arg\min_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{i=1}^{n} \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2 + \frac{n}{\sqrt{2\pi\sigma^2}} \\
&= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{x}_i^\top \mathbf{w} - y_i\right)^2
\end{aligned}$$

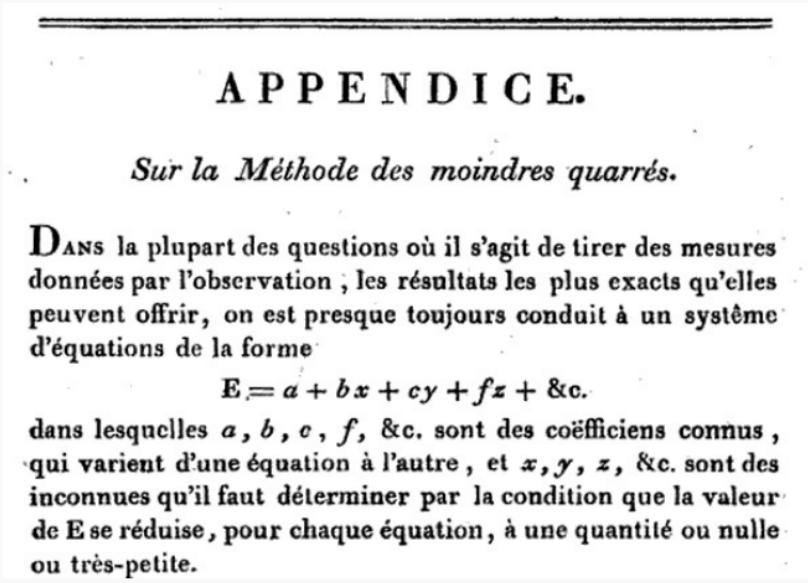
Maximum Likelihood

Thus we've shown that the Least Square estimate is in fact the Maximum Likelihood solution if the error is assumed to be Gaussian.

Historical Notes

Origins of Least Squares

The least-squares method has its origins in the methods of calculating orbits of celestial bodies. It is often credited to Carl Friedrich Gauss (1809) but it was first published by Adrien-Marie Legendre in 1805. The priority dispute comes from Gauss's claim to have used least squares since 1795.



APPENDICE.

Sur la Méthode des moindres quarrés.

DANS la plupart des questions où il s'agit de tirer des mesures données par l'observation, les résultats les plus exacts qu'elles peuvent offrir, on est presque toujours conduit à un système d'équations de la forme

$$E = a + bx + cy + fz + \&c.$$

dans lesquelles $a, b, c, f,$ &c. sont des coëfficiens connus, qui varient d'une équation à l'autre, et $x, y, z,$ &c. sont des inconnues qu'il faut déterminer par la condition que la valeur de E se réduise, pour chaque équation, à une quantité ou nulle ou très-petite.

Legendre (1805), *Nouvelles méthodes pour la détermination des orbites des comètes.*

Origins of Regression

The term **Regression** comes from the publication by Francis Galton *Regression towards mediocrity in hereditary stature (1886).*

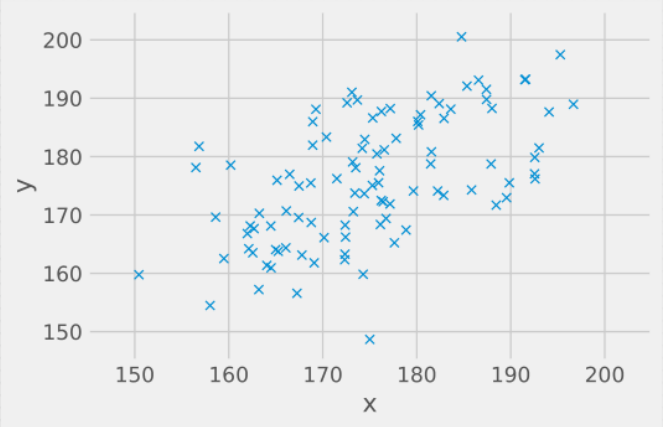
Galton was looking comparing the distribution of heights from parent and their offsprings. He applied Least Squares and observed that his linear fit predicted that parents who are at the tails of the height distribution have offsprings that are closer to the mean of the distribution, eg. taller than average parents are predicted to have shorter offsprings.

Hence the expression **"regression"** towards the mean.

In fact this is a fallacy and a misuse of Least Squares.

Origins of Regression

This is what the scatter plot of the parent's height x vs. the offspring's height y looks like:



The problem is that both measured heights x and y are noisy measurements of some true underlying "height genes" u and v.

The underlying linear model $v = w_1 u + w_0$ becomes:

$$y + \epsilon = w_1(x + v) + w_0$$

where ϵ and v are our noise variables.

What is the impact of having noisy features $x + \nu$ on the normal equations?

$$\mathbf{X'}^\top \mathbf{X'} = \begin{pmatrix} \sum_{i=1}^n 1 & \sum_{i=1}^n x_i + \nu_i \\ \sum_{i=1}^n x_i + \nu_i & \sum_{i=1}^n (x_i + \nu_i)^2 \end{pmatrix} \quad , \quad \mathbf{X'}^\top \mathbf{y} = \begin{pmatrix} \sum_{i=1}^n y_i \\ \sum_{i=1}^n (x_i + \nu_i) y_i \end{pmatrix}$$

assuming that ν is independent of x, we have:
$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^n \nu_i = 0$, $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^n \nu_i x_i = 0$, $\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^n \nu_i y_i = 0$

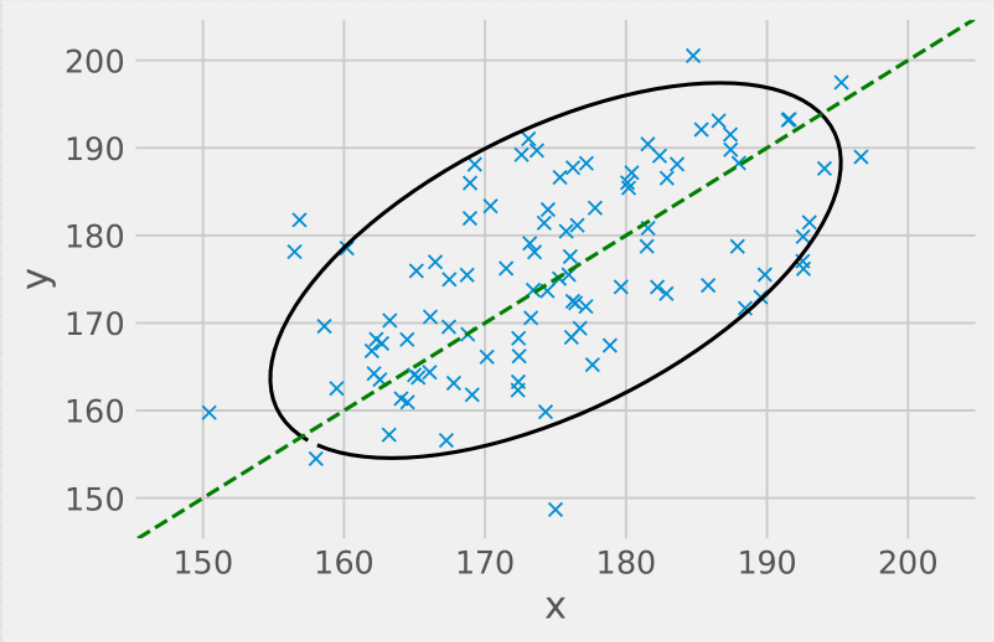
Thus

$$\mathbf{w} \approx \left(\mathbf{X}^\top \mathbf{X} + n \begin{pmatrix} 0 & 0 \\ 0 & \sigma_\nu^2 \end{pmatrix} \right)^{-1} \mathbf{X}^\top \mathbf{y}$$

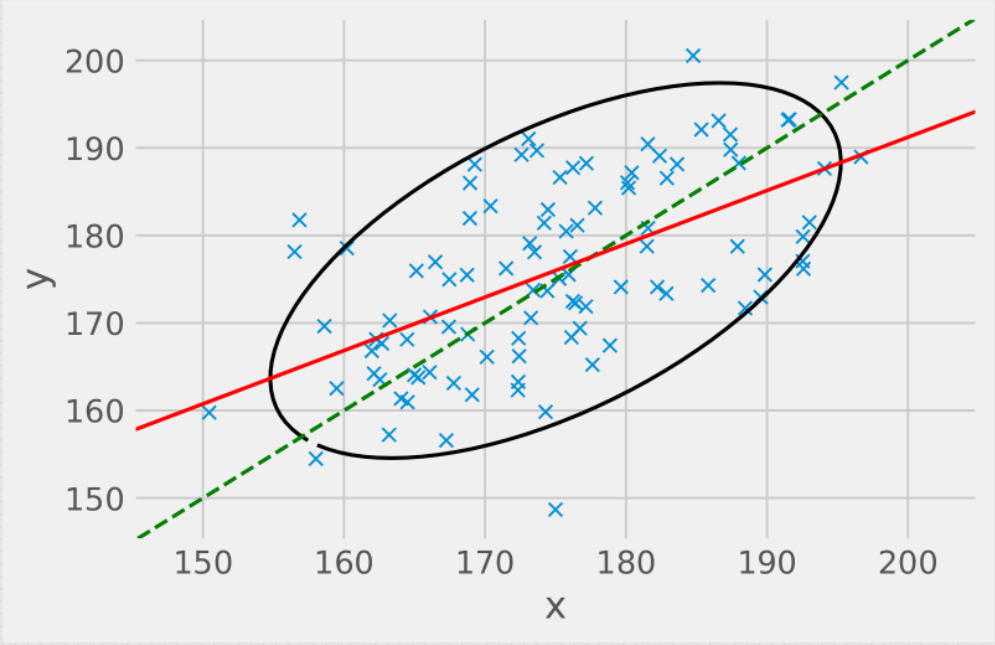
with $\sigma_\nu^2 = \frac{1}{n} \sum_i \nu_i^2$ the noise variance.

This is similar to what we've seen with the Tikhonov regularisation. The noise ν biases w_1 towards 0.

This is the Ground-Truth of the linear model:



but this is what you get with a LS estimate:



Conclusion: if you your features are noisy, then LS will bias your parameter estimation towards 0.

Always check your model. LS assumes $y = \mathbf{x}^\mathsf{T}\mathbf{w} + \epsilon$ with ϵ Gaussian. If your model is $y = (\mathbf{x} + \nu)^\mathsf{T}\mathbf{w} + \epsilon$, it's not the same.

Take Away

We start from a collection of n examples $(\mathbf{x}_i, y_i)_i$. Each of the examples was made up of a number p of features $\mathbf{x}_i = (x_1, \cdots, x_p)$.

We assume that the output can be predicted by a linear model: $y_i = \mathbf{x}_i^\mathsf{T}\mathbf{w} + \varepsilon_i$, with some error ε_i.

We combine all the errors term into a loss function, which is set to be the mean squared error of ε.

The parameters $\hat{\mathbf{w}}$ that minimise the loss function can be derived with the normal equations.

Least square estimation is equivalent is the maximum likelihood solution when we assume that ε follows a Gaussian distribution.

Two issues arise when solving for the LS estimate: underfitting and overfitting. One way to combat overfitting is to use more data and/or to use regularisation.