UNSW AUSTRALIA

Regression

Never Stand Still

COMP9417: Machine Learning & Data Mining

Administration

- Lecturer in Charge:
  - Dr. Gelareh Mohammadi
- Course Admin:
  - Omar Ghattas
- Tutors:
  - Tetian Madfouni
  - Duke Nguyen
  - Lihua Wang
  - Peiyu Tang
  - Jaden Ly
  - Vivien Xian
  - Malhar Patel
  - Rui Tong
  - Jeffrey Meng
  - Zahra Donyavi
  - Daniil Soroka

Communication

- Course website: Moodle
- Email: cs9417@cse.unsw.edu.au (use your UNSW email for all correspondence)
- Moodle forums
- Consult with your tutor in the time slots
- Consultation

Lectures & Tutorials

- Tutorials start in week 2 (No tutorials this week)
- There is no lectures or tutorials in week 6 (term break)
- Attend your allocated tutorial session (some are online and some are in-person)
- The assignments and homework are in Python
- References: a list is provided in the course outline
- Note: Lecture slides do not cover every details we go through in the classroom

Tutorials

- Most weeks will have both lab and tutorial activities (theory and empirical activities).
- No recordings for the tutorials.
- You are supposed to go through the tutorial and lab activities before attending the tutorial class. Your tutor will cover the more challenging questions and there will be time for you to ask your questions

Assessments

- Review exercise (self-assessment) (0%)
- Homework 1 (7.5%)
  - Due in week 4
- Homework 2 (7.5%)
  - Due in week 7
- Assignment (group project) (30%)
  - Due in week 9 and week 10
- Final Exam (55%)

Late submission penalties will apply to all assessable works.

All submissions will be checked for plagiarism.

Final Project

Pick your problem and find the data set:

- This can be done as part of a competition sponsored by an external company which gives prizes to the best projects
- More information will be released soon

- Groups of 4-5 students
- No need to be in the same tutorial
- You need to finalize your groups by **Friday of Week 4**

A Brief Course Introduction

Overview

This course will introduce you to machine learning, covering some of the core ideas, methods and theory currently used and understood by practitioners, including, but not limited to:

- Categories of learning (supervised / unsupervised learning, Neural Nets, etc.)
- Widely-used machine learning techniques and algorithms
- Parametric vs. non-parametric approaches
- Generalisation in machine learning
- Training, validation and testing phases in applications
- Learning Theory

What we will cover

- Core algorithms and model types in machine learning
- Foundational concepts regarding learning from data
- Relevant theory to inform and generalise understanding
- Practical applications

What we will NOT cover

- Probability and statistics (basic knowledge is expected)
- Lots of neural nets and deep learning
- Reinforcement learning
- Big data
- Ethical aspects of AI and ML

Although all of these are interesting and important topics!

Some definitions

The field of machine learning is concerned with the question of how to construct computer programs that automatically improve from experience.

“Machine Learning”. T. Mitchell (1997)

Machine learning, then, is about making computers modify or adapt their actions (whether these actions are making predictions, or controlling a robot) so that these actions get more accurate, where accuracy is measured by how well the chosen actions reflect the correct ones.

“Machine Learning”. S. Marsland (2015)

Some definitions

The term machine learning refers to the automated detection of meaningful patterns in data.

“Understanding Machine Learning”. S. Shwartz and S. David (2014)

Data mining is the extraction of implicit, previously unknown, and potentially useful information from data.

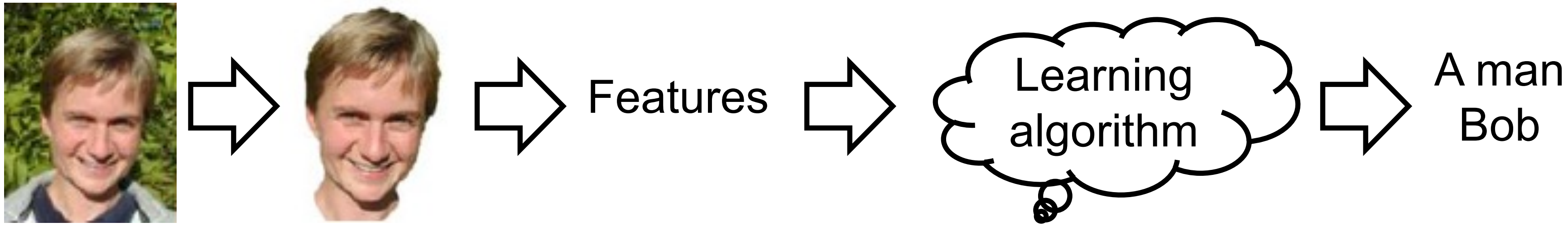
“Data Mining”. I. Witten et al. (2016)

Examples

- Object recognition / object classification
- Text classification (e.g., spam/non-spam emails)
- Speech recognition
- Event detection
- Recommender systems
- Human behaviour recognition (emotions, state of mind, etc.)
- Automatic medical diagnosis

Example

Simple face recognition pipeline:

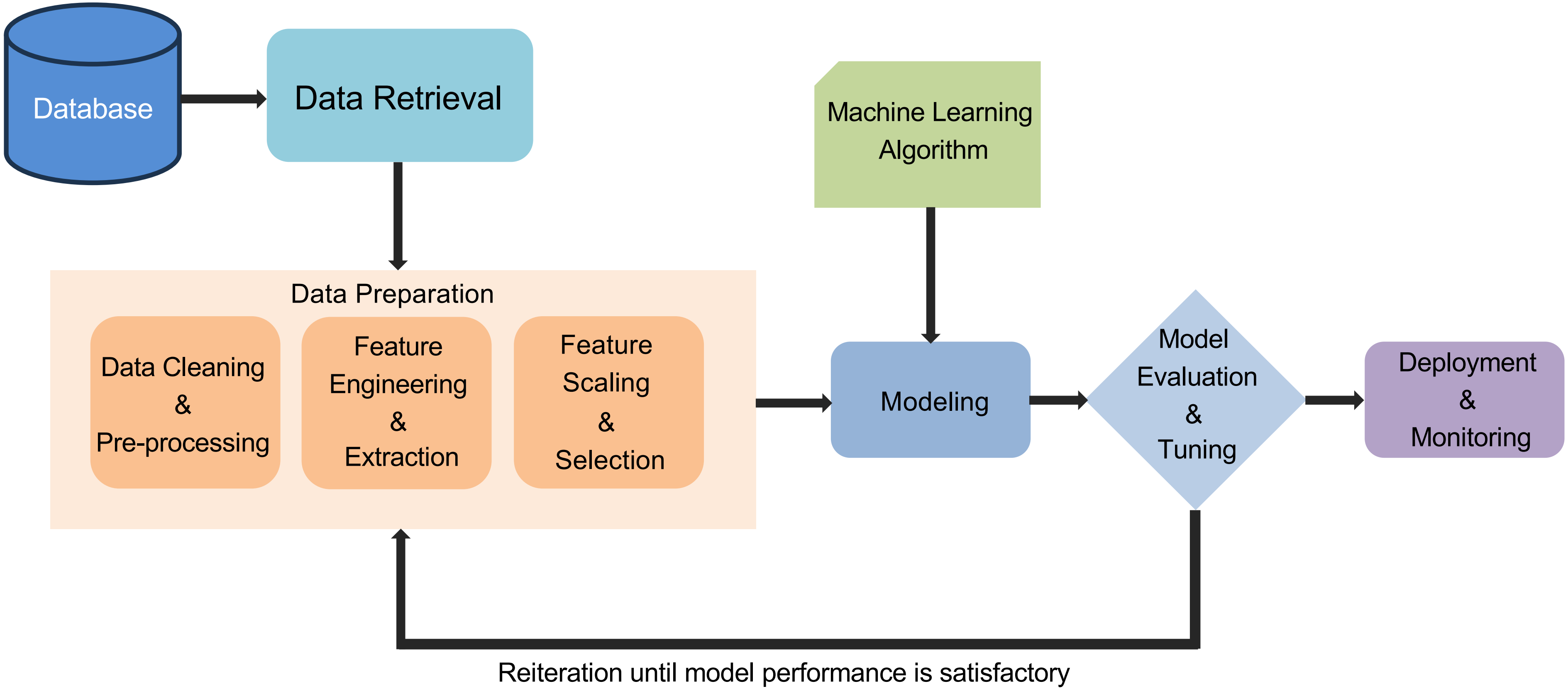


Features

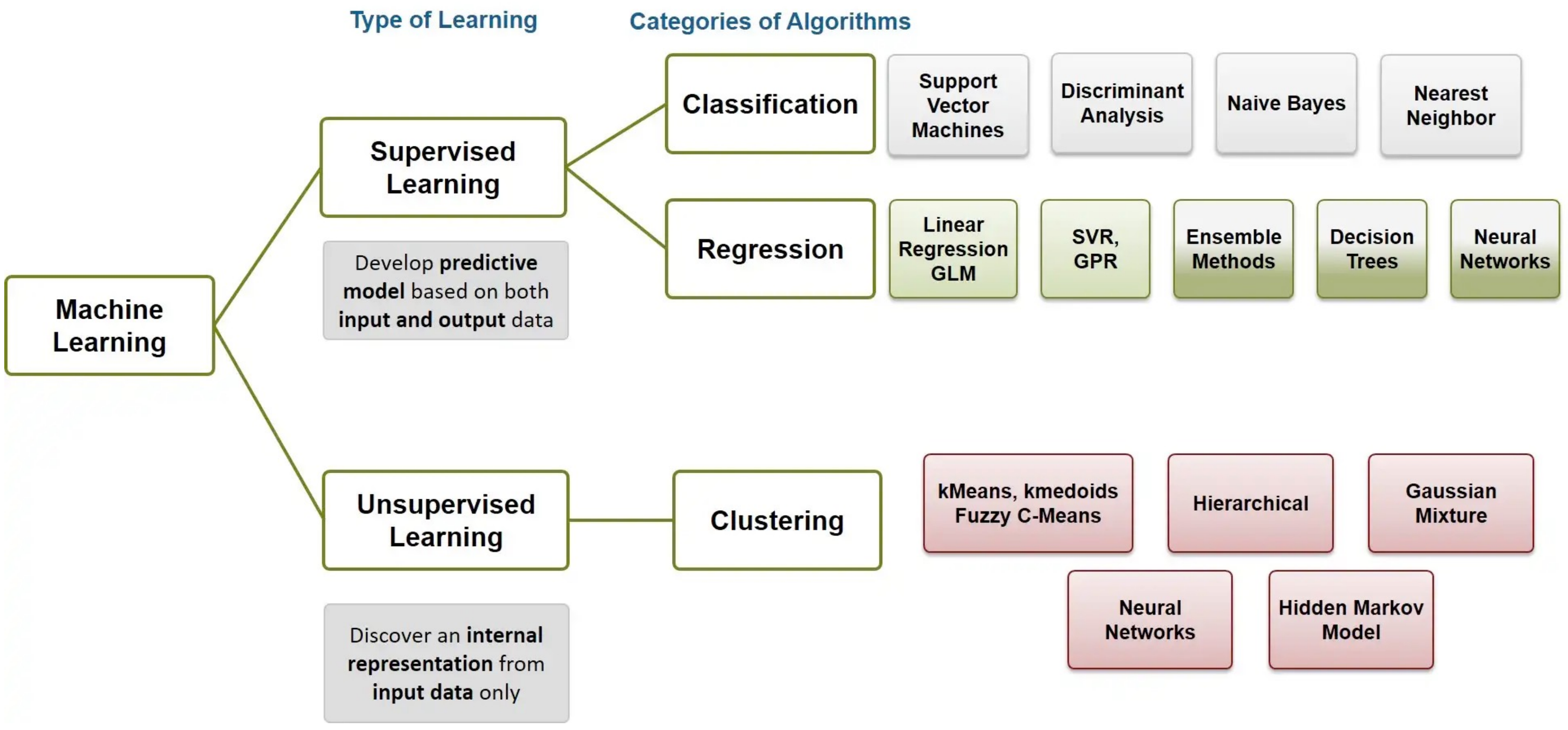
Learning algorithm

A man
Bob

Machine learning pipeline


Database
Data Retrieval
Machine Learning Algorithm
Data Preparation
Data Cleaning & Pre-processing
Feature Engineering & Extraction
Feature Scaling & Selection
Modeling
Model Evaluation & Tuning
Deployment & Monitoring
Reiteration until model performance is satisfactory

Supervised and unsupervised learning



Type of Learning

Categories of Algorithms

Machine Learning

Supervised Learning

Develop **predictive model** based on both **input and output** data

Classification

Support Vector Machines

Discriminant Analysis

Naive Bayes

Nearest Neighbor

Regression

Linear Regression GLM

SVR, GPR

Ensemble Methods

Decision Trees

Neural Networks

Unsupervised Learning

Discover an **internal representation** from **input data** only

Clustering

kMeans, kmedoids Fuzzy C-Means

Hierarchical

Gaussian Mixture

Neural Networks

Hidden Markov Model

Supervised and unsupervised learning

The most widely used categories of machine learning algorithms are:

- Supervised learning – output class (or label) is given
- Unsupervised learning – no output class is given

There are also hybrids, such as semi-supervised learning, and alternative strategies to acquire data, such as reinforcement learning and active learning.

Note: output class can be real-valued or discrete, scalar, vector, or other structure . . .

Supervised and unsupervised learning

Supervised learning tends to dominate in applications.

Why ?

Supervised and unsupervised learning

Supervised learning tends to dominate in applications because generally, it is much easier to define the problem and develop an error measure (loss function) to evaluate different algorithms, parameter settings, data transformations, etc. for supervised learning than for unsupervised learning.

Supervised and unsupervised learning

Unfortunately . . .

In the real world it is often difficult to obtain good labelled data in sufficient quantities

So, in such cases unsupervised learning is really what you want . . .

but currently, finding good unsupervised learning algorithms for complex machine learning tasks remains a research challenge.

Aims

This lecture (which is on regression) will introduce you to machine learning approaches to the problem of numerical prediction. Following it you should be able to reproduce theoretical results, outline algorithmic techniques and describe practical applications for the topics:

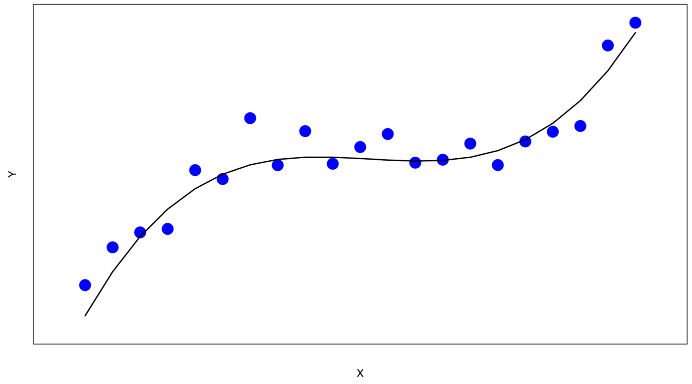
- supervised learning task of numeric prediction
- how linear regression solves the problem of numeric prediction
- fitting linear regression by least squares error criterion
- non-linear regression via linear-in-the-parameters models
- parameter estimation for regression
- local (nearest-neighbour) regression

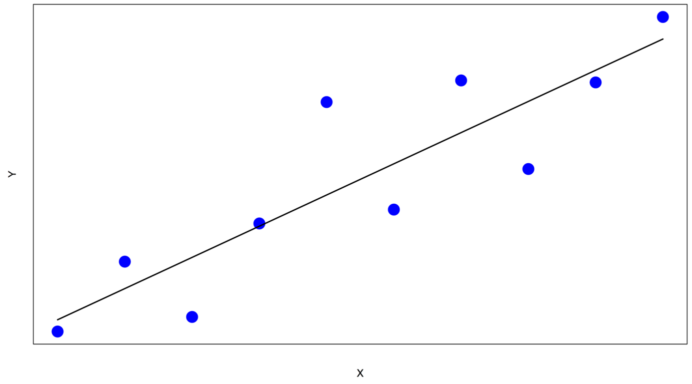
Introduction to Regression

Introduction to Regression

The “most typical” machine learning approach is to apply supervised learning methods for classification, where the task is to learn a model to predict a discrete value for data instances . . .

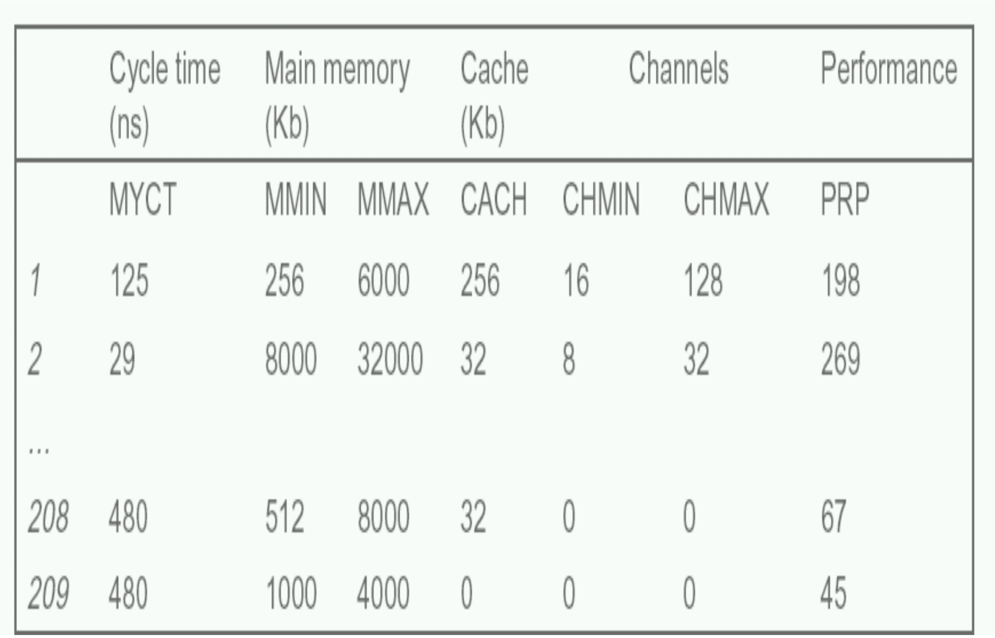
. . . however, we often find tasks where the most natural representation is that of prediction of numeric values. So, we need regression in those tasks





Introduction to Regression

Example – task is to learn a model to **predict CPU performance** from a dataset of example of 209 different computer configurations:



	Cycle time (ns)	Main memory (Kb)		Cache (Kb)	Channels		Performance
	MYCT	MMIN	MMAX	CACH	CHMIN	CHMAX	PRP
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
…							
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

Introduction to Regression

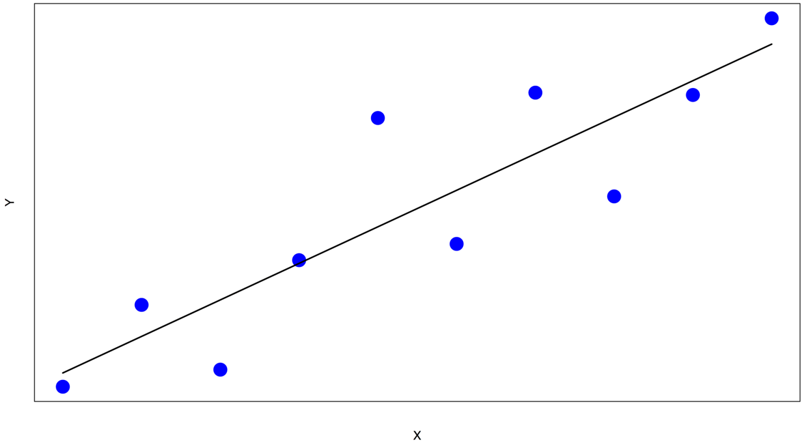
Result: a linear regression equation fitted to the CPU dataset.

$$\begin{aligned} PRP \quad = \quad & -\ 56.1 \\ & +\ 0.049\ MYCT \\ & +\ 0.015\ MMIN \\ & +\ 0.006\ MMAX \\ & +\ 0.630\ CACH \\ & -\ 0.270\ CHMIN \\ & +\ 1.46\ CHMAX \end{aligned}$$

Learning Linear Regression Models

Linear Regression

In linear regression, we assume there is a linear relationship between the output and feature(s)/input variable(s); this means the expected value of the output given an input, $E[y|x]$ is linear in input x



Example with one variable: $\hat{y} = bx + c$

Problem: given the data, estimate b and c

Linear regression

- Training instances: $\langle x_j, y_j \rangle, j = 1, \ldots, m$
- x_j is an input vector
- y_j is the output to be predicted
- Each input has n attributes/features $x_j = [x_{j1}, x_{j2}, \ldots, x_{jn}]^T$
- x and y are a general observation with $x = [x_1, x_2, \ldots, x_n]^T$
- $\theta = [\theta_0, \theta_1, \ldots, \theta_n]^T$
- $\mathbb{y}$ is the vector of outputs:$\mathbb{y} = [y_1, \ldots, y_m]^T$
- X is the matrix of inputs where each row is one observation

Linear regression

- Linear regression can be used for numeric prediction from numeric attributes
- In linear models, the outcome is a **linear combination of attributes**:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n = h(x)$$

- **Weights** are estimated **from the observed data** (training data)
- **Predicted value** for the first training instance x_i is:

$$\hat{y}_1 = \theta_0 + \theta_1 x_{11} + \theta_2 x_{12} + \cdots + \theta_n x_{1n} = \sum_{i=0}^{n} \theta_i x_{1i} = x_1^T \theta = h(x_1)$$

To simplify the notation, we set $x_0 = 1$ and define $x = [x_0, x_1, \dots, x_n]^T$

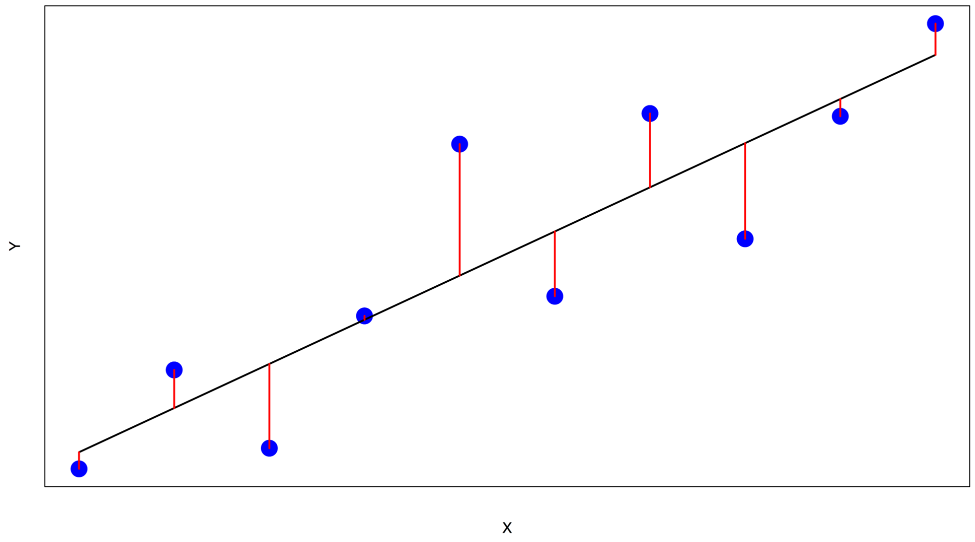
Linear regression

- **Univariate regression**: one input variable/feature/attribute is used to predict one output variable
- **Multiple regression / multivariable regression**: more than one variables/features/attributes are used to predict one output variable

Linear Regression

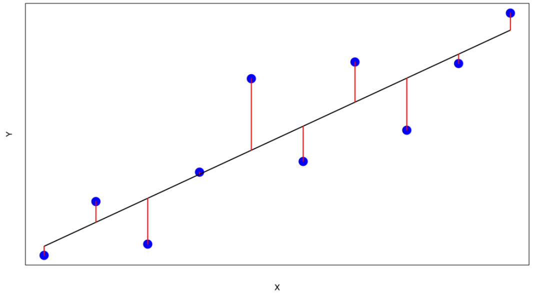
Infinite number of lines can be fitted, depending on how we define the best fit criteria

…but the most popular estimation model is “Least Squares”, also known as “Ordinary Least Squares” (OLS) regression



Minimizing Error

- Error = Difference between the predicted value and the actual value



- We want to minimize the error over all samples!
- We define the total error as the sum of squared errors and searching for $n+1$ parameters to minimize that
- Sum of squared error for m samples/observations

$$J(\theta)=\sum_{j=1}^{m}(y_j-\sum_{i=0}^{n}\theta_i x_{ji})^2=\sum_{j=1}^{m}(y_j-x_j^T\theta)^2=(\mathbb{y}-X\theta)^T(\mathbb{y}-X\theta)$$

$J(\theta)$ is called cost function or loss function. This concept generalizes to all functions that measure the distance between the predicted and true output. (in OLS framework, it is also called Residual Sum of Squares (RSS))

Minimizing Squared Error (normal equations)

Here is how matrix X and vector $\mathbb{y}$ look like:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \dots & x_{1n} \\ 1 & x_{21} & x_{22} & \dots & x_{2n} \\ & & \vdots & & \\ 1 & x_{m1} & x_{m2} & \dots & x_{mn} \end{bmatrix} \qquad \mathbb{y} = \begin{bmatrix} y_1 \\ y_2 \\ \\ y_m \end{bmatrix}$$

You have to be careful to **add a column of ones** in X to count for the intercept parameter.

Minimizing Squared Error

This cost function can be also written as:

$$J(\theta) = \frac{1}{m}\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$$

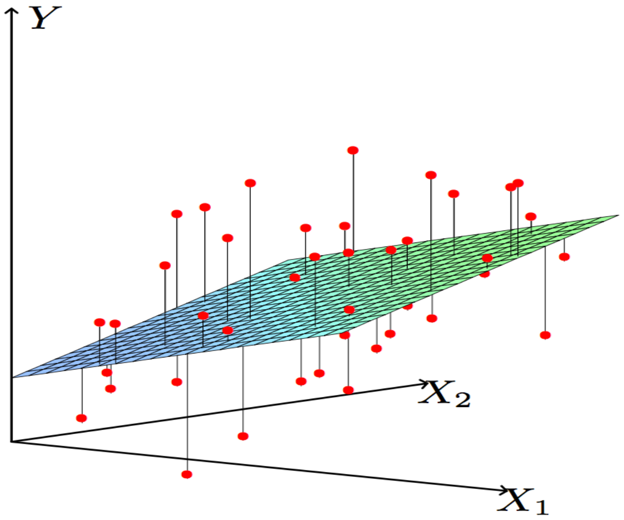
Which is the average of squared error and minimizing it, leads to the same θ, that we get with $J(\theta)$ without taking the average.

The $\frac{1}{m}\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$ is called mean squared error or briefly MSE.

Minimizing Squared Error

Multiple regression example:

Given 2 variables/attributes with real values $x_1, x_2 \in \mathbb{R}$, labelled with a real-valued variable y, find "plane of best fit" that captures the dependency of y on x_1 and x_2.



Again we can use MSE to find the best plane that fits the data.

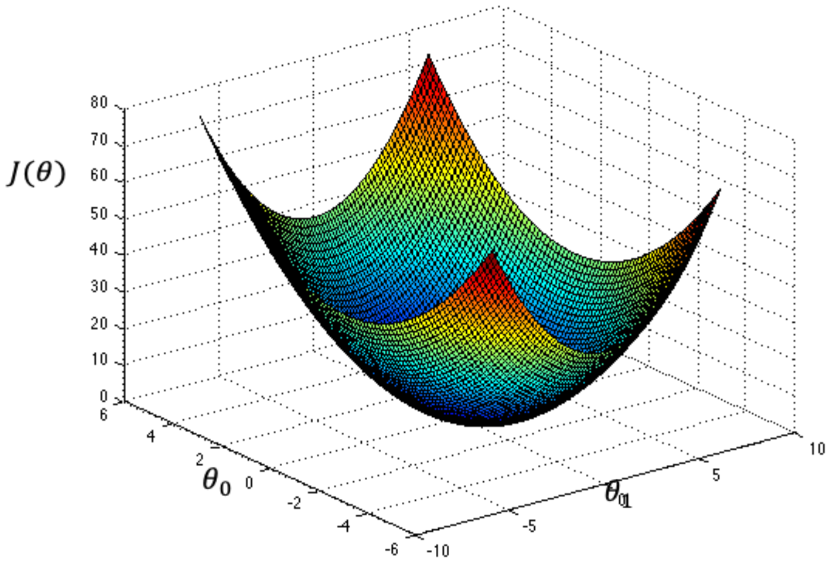
For more than two variables, we find the best hyper-plane.

Least Squares Regression

Question: How to estimate the parameters such to minimize the cost function $J(\theta)$?

If you compute $J(\theta)$ for different values of θ in linear regression problem, you will end up with a convex function which in this particular problem has one global minima.

…and we can use a search algorithm which starts with some “initial guess” for θ and repeatedly changes θ to make $J(\theta)$ smaller, until it converge to minimum value.
One of such algorithms is gradient descent



Gradient Descent

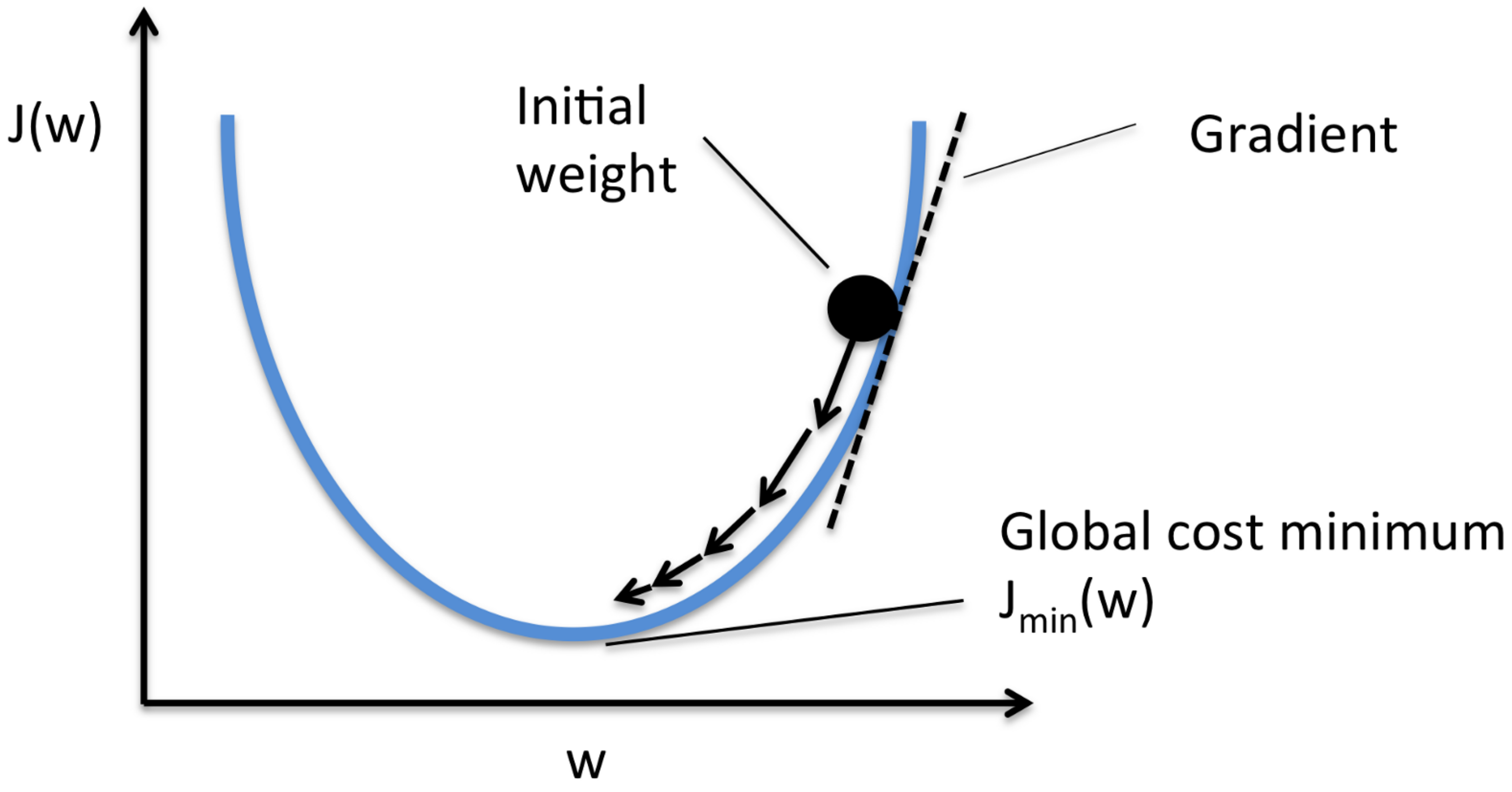
Gradient descent starts with some initial θ, and repeatedly performs an update:

$$\theta_i^{(t+1)} \coloneqq \theta_i^{(t)} - \alpha \frac{\partial}{\partial \theta_i} J(\theta_i^{(t)})$$

α is the learning rate (a.k.a. step size). This algorithm takes a step in the direction of the steepest decrease in $J(\theta)$

To implement this algorithm, we have to work out the partial derivative term for $J(\theta) = \frac{1}{m}\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$

Gradient Descent



From: Gradient Descent: All you need to know, by S. Suryansh

Gradient Descent

If we focus on **only one sample** out of m samples, then the cost function is:

$$J(\theta) = (y_j - h_\theta(x_j))^2 = (y_j - \sum_{i=1}^{n} x_{ji}\theta_i)^2$$

$$h_\theta(x_j) = \sum_{i=1}^{n} x_{ji}\theta_i = x_j^T\theta$$

Taking the derivative will give:

$$\frac{\partial}{\partial\theta_i} J(\theta) = -2(y_j - h_\theta(x_j))x_{ji}$$

So, for a **single** training sample, the update rule is:

$$\theta_i \coloneqq \theta_i + 2\alpha(y_j - h_\theta(x_j))x_{ji} \qquad \text{(for every } i\text{)}$$

The update rule for squared distance is called "**Least Mean Squares**" (LMS), also known as **Widrow-Hoff**

Gradient Descent

We looked at LMS rule for only a single sample. But what about other samples? There are two ways: Batch Gradient Descent and Stochastic Gradient Descent. For the following cost/loss function:

$$J(\theta) = \frac{1}{m}\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$$

1. **Batch Gradient Descent:**

$$\theta_i^{(t+1)} = \theta_i^{(t)} + \alpha\frac{2}{m}\sum_{j=1}^{m}(y_j - h_{\theta^{(t)}}\left(x_j\right))x_{ji} \text{ (for every } i\text{)}$$

Replace the gradient with the sum of gradient for all samples and continue until convergence.

Convergence means that, the estimated θ will be stabilized.

Gradient Descent

2. Stochastic Gradient Descent:

```
$for\ j = 1\ \ to\ \ m\ \{$
    $\theta_i = \theta_i + 2\alpha(y_j - h_\theta(x_j))x_{ji}$   (for every $i$)
$\}$
```

Repeat this algorithm until convergence.

What this algorithm does?

Gradient Descent

In stochastic gradient descent, θ gets updated at any sample separately. This algorithm is much less costly than batch gradient descent, however it may never converge to the minimum.

Gradient Descent

In both algorithms:

- You can start with an arbitrary (random) values for your parameters θ_i (initialization)
- You have to be careful with the selection of learning rate, α, as a small value can make the algorithm very slow, and a big value may stop the algorithm from convergence

Minimizing Squared Error (normal equations)

Gradient descent is one way of minimizing our cost function $J(\theta)$ which is an iterative algorithm.

But maybe you can find the minimum of $J(\theta)$ explicitly by taking its derivatives and setting them to zero. This is also called exact or closed-form solution:

$$\frac{\partial}{\partial\theta}J(\theta) = 0$$

$$J(\theta) = (\mathbb{y} - X\theta)^T(\mathbb{y} - X\theta)$$

$$\frac{\partial}{\partial\theta}J(\theta) = -2X^T(\mathbb{y} - X\theta) = 0$$

$$X^T(\mathbb{y} - X\theta) = 0$$

$$\boxed{\theta = (X^T X)^{-1}X^T\mathbb{y}}$$

Minimizing Squared Error (probabilistic interpretation)

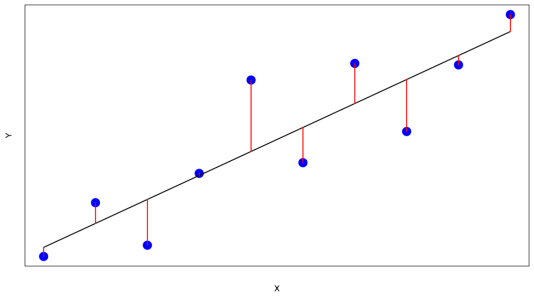
We can write the relationship between input variable x and output variable y as:

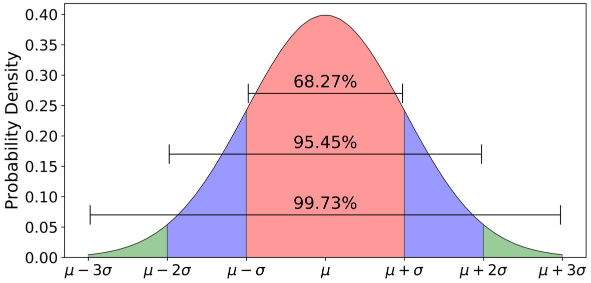
$$y_j = x_j^T \theta + \varepsilon_j$$

And ε_j is an error term which might be unmodeled effect or random noise. Let’s assume ε_js are independent and identically distributed ($i.i.d.$) according to a Gaussian distribution:

$$\varepsilon_j \sim N(0, \sigma^2)$$

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





Minimizing Squared Error (probabilistic interpretation)

This implies that:

$$p(\varepsilon_j) = p(y_j|x_j;\theta) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(y_j - x_j^T\theta)^2}{2\sigma^2})$$

So we want to estimate θ such that we maximize the probability of output y given input x over all m training samples:

$\mathcal{L}(\theta) = p(\mathbb{y}|X;\theta)$ (this is called **Likelihood** function)

Minimizing Squared Error (probabilistic interpretation)

Since we assumed independence over ε_j, that implicitly means that our training samples are also independent from each other. Based on this assumption, we can write $\mathcal{L}(\theta)$ as follows:

$$\mathcal{L}(\theta) = \prod_{j=1}^{m} p(y_j | x_j; \theta)$$

$$= \prod_{j=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y_j - x_j^T\theta)^2}{2\sigma^2})$$

Now, we have to estimate θ such that it maximized $\mathcal{L}(\theta)$ to have as high probability as possible. This is called **maximum likelihood**.

Minimizing Squared Error (probabilistic interpretation)

We know (from math) that to find θ that maximized $\mathcal{L}(\theta)$, we can also maximize any strictly increasing function of $\mathcal{L}(\theta)$. In this case it would be easier if we **maximize the log likelihood** $\ell(\theta)$:

$$\ell(\theta) = \log \mathcal{L}(\theta) = \log \prod_{j=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(y_j - x_j^T\theta)^2}{2\sigma^2}) =$$

$$= m \log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \frac{1}{\sigma^2}\frac{1}{2}\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$$

So, maximizing $\ell(\theta)$ is equal to minimizing $\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$

Do you know what is $\sum_{j=1}^{m}(y_j - x_j^T\theta)^2$?

Minimizing Squared Error (probabilistic interpretation)

- This simply shows that under certain assumptions ($\varepsilon_j \sim N(0, \sigma^2)$) and $i.i.d.$) the least-squared regression is equivalent to find maximum likelihood estimate of θ.

- Note that the value of σ^2 does not affect the choice of θ

Linear Regression Assumptions

- **Linearity**: The relationship between x and the mean of y is linear.
- **Homoscedasticity**: The variance of residual is the same for any value of x.
- **Independence**: Observations are independent of each other.
- **Normality**: For any fixed value of x, y is normally distributed.

Step back: Statistical Techniques for Data Analysis

Probability vs Statistics: The Difference

Probability versus Statistics

- **Probability**: reasons from populations to samples
  - This is deductive reasoning, and is usually sound (in the logical sense of the word)
- **Statistics**: reasons from samples to populations
  - This is inductive reasoning, and is usually unsound (in the logical sense of the word)

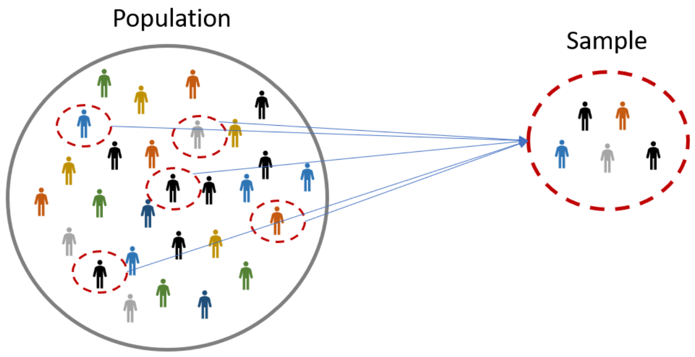
Sampling

Where do the Data come from? (Sampling)

- For groups (populations) that are fairly homogeneous, we do not need to collect a lot of data. (We do not need to sip a cup of tea several times to decide that it is too hot.)
- For populations which have irregularities, we will need to either take measurements of the entire group, or find some way of get a good idea of the population without having to do so
- *Sampling* is a way to draw conclusions about the population without having to measure all of the population. The conclusions need not be completely accurate.
- All this is possible if the sample closely resembles the population about which we are trying to draw some conclusions

What We Want From a Sampling Method

- No systematic bias, or at least no bias that we cannot account for in our calculations
- The chance of obtaining an unrepresentative sample can be calculated. (So, if this chance is high, we can choose not to draw any conclusions.)
- The chance of obtaining an unrepresentative sample decreases with the size of the sample



Estimation

Estimation from a Sample

- In statistics, estimation refers to the process by which one makes inferences about a population, based on information obtained from a sample.
- Estimating some aspect of the population using a sample is a common task. E.g. sample mean are used to estimate population means
- Along with the estimate, we also want to have some idea of the accuracy of the estimate (usually expressed in terms of confidence limits)
- Some measures calculated from the sample are very good estimates of corresponding population values. For example, the sample mean m is a very good estimate of the population mean μ. But this is not always the case. For example, the range of a sample usually underestimates the range of the population

Estimation from a Sample

- We will have to clarify what is meant by a “good estimate”. One meaning is that an estimator is correct on average. For example, on average, the mean of a sample is a good estimator of the mean of the population
- For example, when a number of sample sets are drawn and the mean of each is found, then average of these means is equal to the population mean
- Such an estimator is said to be *statistically unbiased*

Estimate of the mean and Variance

Mean:

- The arithmetic mean of a sample is $m = \frac{1}{N}\sum_{i=1}^{N} x_i$, where the observations are $x_1, x_2, \ldots, x_N$
- This works very well when the data follow a "normal" distribution
- If we can group the data so that observation x_1 occurs f_1 times, x_2 occurs f_2 times and so on, then the mean is calculated even easier as $m = \frac{1}{N}\sum_i x_i f_i$
- If we define relative frequencies as $p_i = \frac{f_i}{N}$, then the mean is simply the observations weighted by relative frequency. That is $m = \sum_i x_i p_i$
- So, the expected value of a discrete random variable X is:

$$E(X) = \sum_i x_i p(X = x_i)$$

Estimates of the Mean and Variance

Variance: measures how far a set of random numbers are spread out from their average value

- Standard deviation (square root of variance) is estimates as:

$$s = \sqrt{\frac{1}{N-1}\sum_{i}(x_i - m)^2}$$

- Again this is a very good estimate when the data are modeled by Normal distribution.
- For grouped data, this is modified to:

$$s = \sqrt{\frac{1}{N-1}\sum_{i}(x_i - m)^2 f_i}$$

- Again, we can write this in terms of expected values, which estimates the scatter (spread) of values of a random variable X around a mean value $E(X)$:

$$Var(X) = E((X - E(X))^2) = E(X^2) - [E(X)]^2$$

Variance is the “mean of squares minus the squares of means”

Covariance and Correlation

Covariance is a measure of relationship between two random variables:

$$cov(x, y) = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{N - 1} = \frac{(\sum_i x_i y_i) \; - N\bar{x}\bar{y}}{N - 1}$$

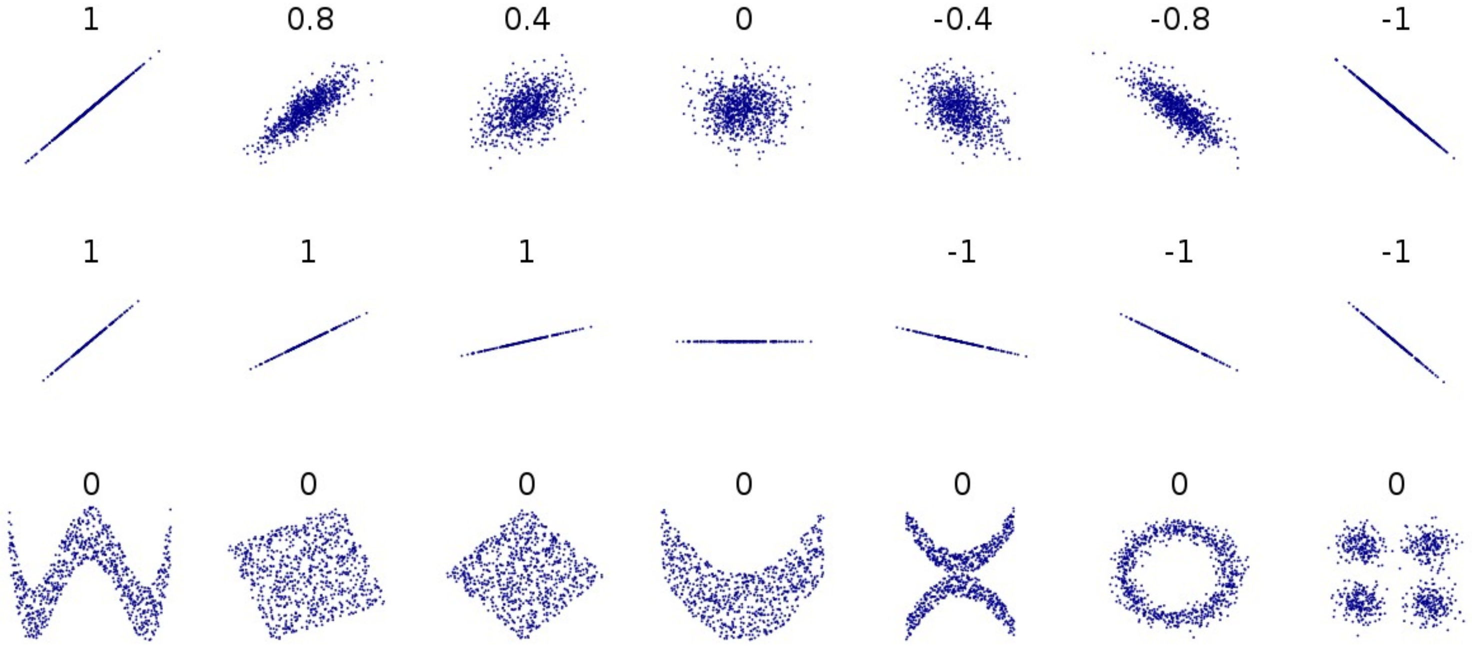
Correlation is a measure to show how strongly a pair of random variables are related

- The formula for computing correlation between x and y is:

$$r = \frac{cov(x,y)}{\sqrt{var(x)}\sqrt{var(y)}}$$

This is also called *Pearson's correlation coefficient*

Correlation



Pearson correlation in some sets of (x,y). [Wikipedia]

Correlation

- The correlation coefficient is a number between -1 and $+1$ that shows whether a pair of variable x and y are associated or not and whether their scatter in the association is high or low:
  - A value close to 1 shows that high values of x are associated with high values of y and low values of x are associates with low values of y, and scatter is low
  - A value near 0 indicates that there is no association and there is a large scatter
  - A value close to -1 suggests a strong inverse association between x and y
- Correlation is only appropriate when x and y are roughly linearly associated (does not work well when the association is curved)

Correlation

- Correlation is a quick way of checking whether there is some linear association between two variables x and y
- The sign of the value tells you the direction of the association
- The correlation does not model any relationship. That is, given a particular x, you can not use r to calculate a y value.
  - It is possible for two datasets to have the same correlations, but different relationships.
  - It is possible for two datasets to have the same relationships, but different correlations

Note:

- Do not use correlations to compare datasets. All you can get from correlation is whether there is a positive or negative relationship between between x and y
- Do not use correlation to imply x causes y or the other way around

Regression

Univariate linear regression: (only one independent variable)

Goal: fit a line such that $\hat{y} = \theta_0 + \theta_1 x$

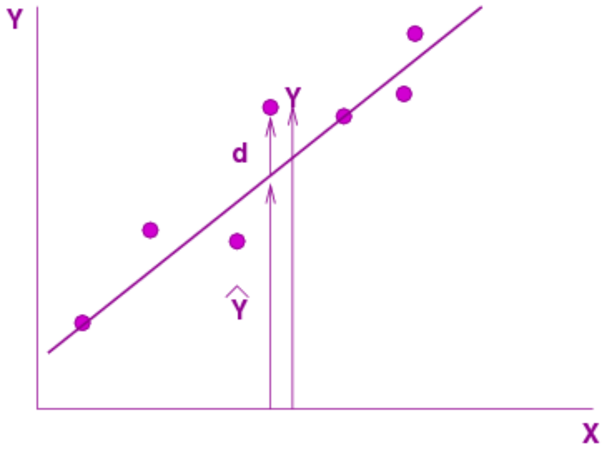
How: minimize $J(\theta) = \sum_j (y_j - \hat{y}_j)^2$ (Least squares estimator)

- If we expand the explicit solution we saw before, we can see:

$$\theta_1 = \frac{cov(x, y)}{var(x)}$$

Where $cov(x, y)$ is the covariance of x and y

- and $\theta_0 = \bar{y} - \theta_1 \bar{x}$



Univariate linear regression

In univariate regression we aim to find the relationship between y and one independent variable x.

Example:

Suppose we want to investigate the relationship between people's height and weight. We collect m height and weight measurements:

$$(h_j, w_j), \qquad j = 1, \dots, m$$

Univariate linear regression assumes a linear relation $\hat{w} = \theta_0 + \theta_1 h$, with parameters θ_0, θ_1 chosen such that the sum of squared residuals $\sum_{j=1}^{m} \left(w_j - \theta_0 + \theta_1 h_j \right)^2$ is minimized

Univariate linear regression

In order to find the parameters we take partial derivatives, set the partial derivatives to 0 and solve for θ_0. As we saw before, this will lead to:

$$\theta_1 = \frac{cov(h,w)}{var(h)} = \frac{\sum_{j=1}^{m}(h_j - \bar{h})(w_j - \bar{w})}{\sum_{j=1}^{m}(h_j - \bar{h})^2}$$

$$\theta_0 = \bar{w} - \theta_1 \bar{h}$$

Linear regression: intuitions

- Adding a constant to all x-values (a translation) will affect only the intercept but not the regression coefficient (since it is defined in terms of deviations from the mean, which are unaffected by a translation).

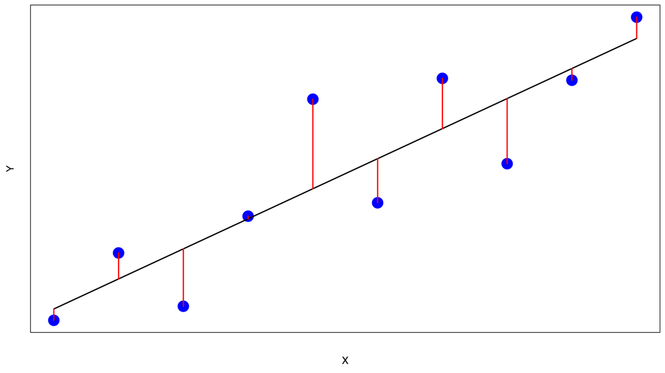
- So we could zero-centre the x -values by subtracting x, in which case the intercept is equal to y .

- We could even subtract y from all y -values to achieve a zero intercept, without changing the problem in an essential way.

Linear regression: intuitions

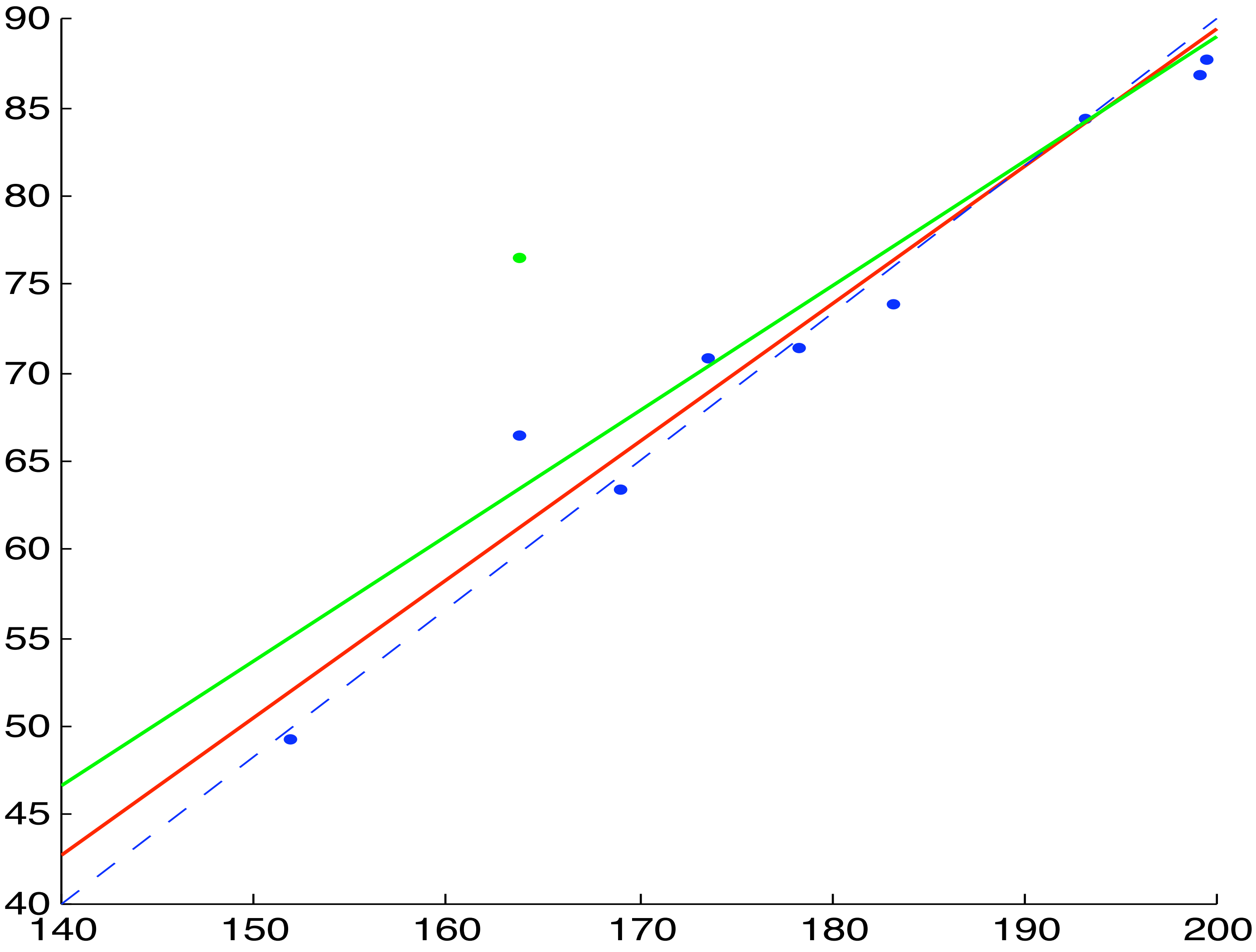
Another important point to note is that the **sum of the residuals** of the least square (or mean squared error) **is zero**

$$\sum_{j=1}^{m} \left(y_j - x_j^T\theta\right) = 0$$



While this property is intuitively appealing it is worth keeping in mind that it also makes linear regression susceptible to *outliers:* points that are far removed from the regression line, often because of the measurement error.

The effect of outliers



The effect of outliers

Shown on previous slide:

- Suppose that, as the result of a transcription error, one of the weight values from the previous example of univariate regression is increased by 10 kg. The diagram shows that this has a considerable effect on the least-squares regression line.
- Specifically, we see that one of the blue points got moved up 10 units to the green point, changing the red regression line to the green line.

Multiple linear regression

Multiple Regression

Often, we are interested in modelling the relationship of y to several other variables. In observational studies the value of y may be affected by the value of several variables.

Example:

Suppose we want to predict people's weight from their height and body frame size (usually measured by wrist size). We collect m height, weight and body frame measurement:

$$(h_j, f_j, w_j), \qquad j = 1, \dots, m$$

Multiple Regression

Similar to before the linear regression model is:

$$\widehat{w} = \theta_0 + \theta_1 h + \theta_2 f,$$

And similar to univariate linear regression, we have to minimize the sum of squared residuals

$$\sum_{j=1}^{m} \left(w_j - (\theta_0 + \theta_1 h_j + \theta_2 f_j)\right)^2$$

- Including more variables can give a narrower confidence interval on the prediction being made
- With many variables, the regression equations and the expressions for the θ are expressed better using a matrix representation(as we used before) for sets of equations

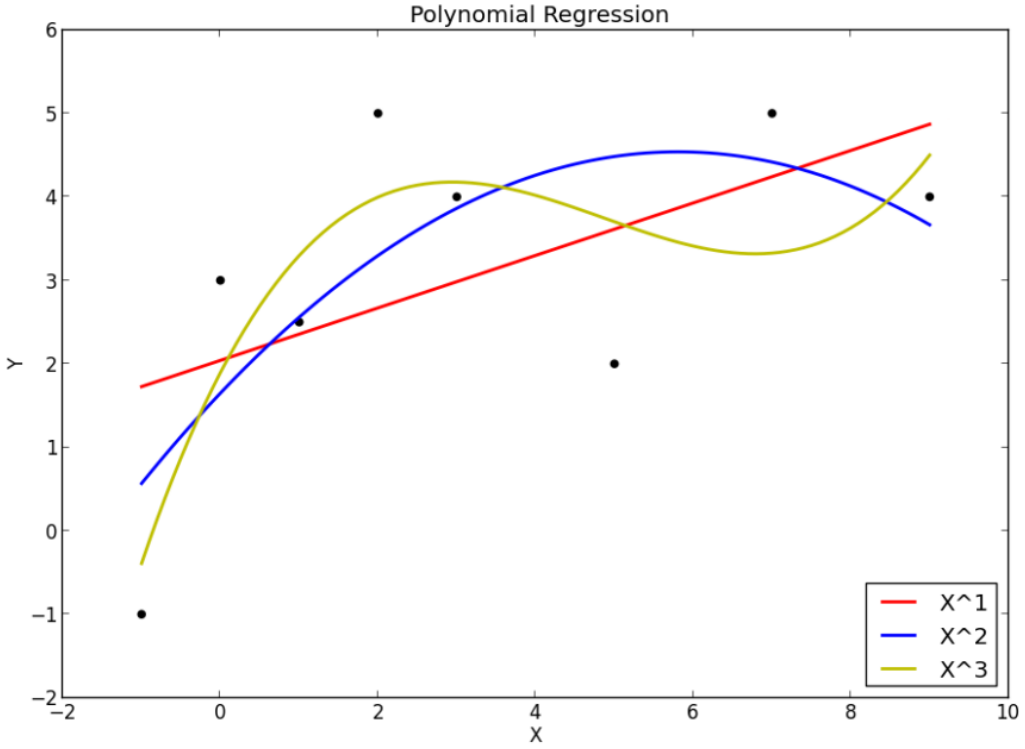
Linear Regression for curve shapes

You may think that linear regression produces straight lines or planes, and nonlinear equations models produce curvature!!

Well, that’s not completely correct. With some tricks we can produce curves with linear regression

Linear Regression for curve shapes

Example



Linear Regression for curve shapes

- In this example we can predict the output with different models:
- $\hat{y} = \theta_0 + \theta_1 x_1$
- $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2,\ x_2 = x_1^2. \rightarrow\ \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- $\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3, x_2 = x_1^2,\ x_3 = x_1^3 \rightarrow\ \hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

As you can see, these nonlinear models can still be treated like linear regression and they can fit curvature. They are still linear in parameters. (Nonlinear regression is not linear in parameters, e.g., $\hat{y} = \frac{\theta_1 x}{\theta_2 + x}$)

Linear Regression for curve shapes

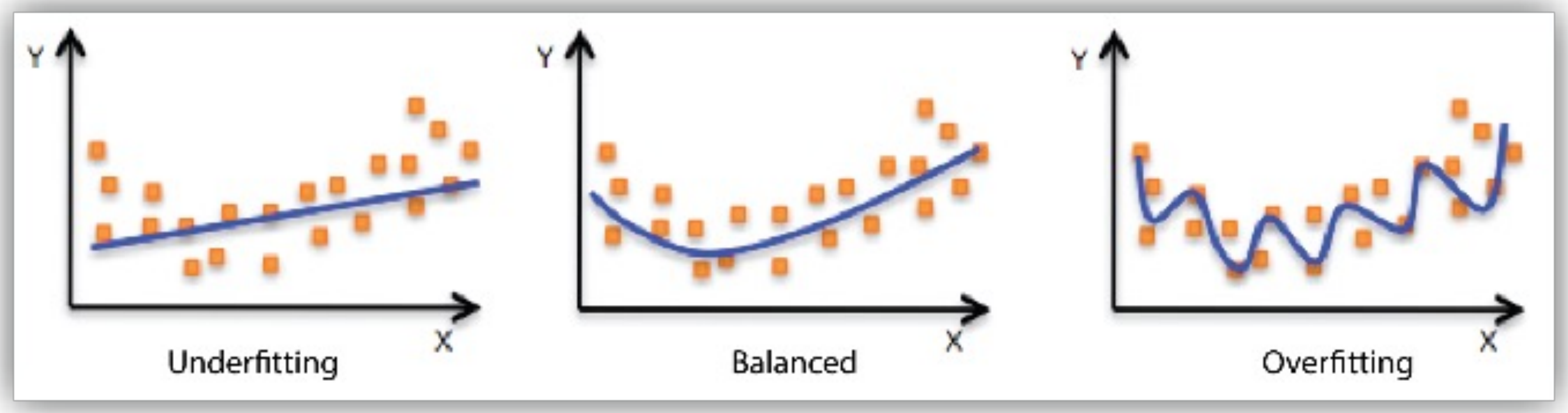


Image from AWS website

Linear Regression for curve shapes

Question:

How to control for degree of complexity of the model to avoid overfitting?

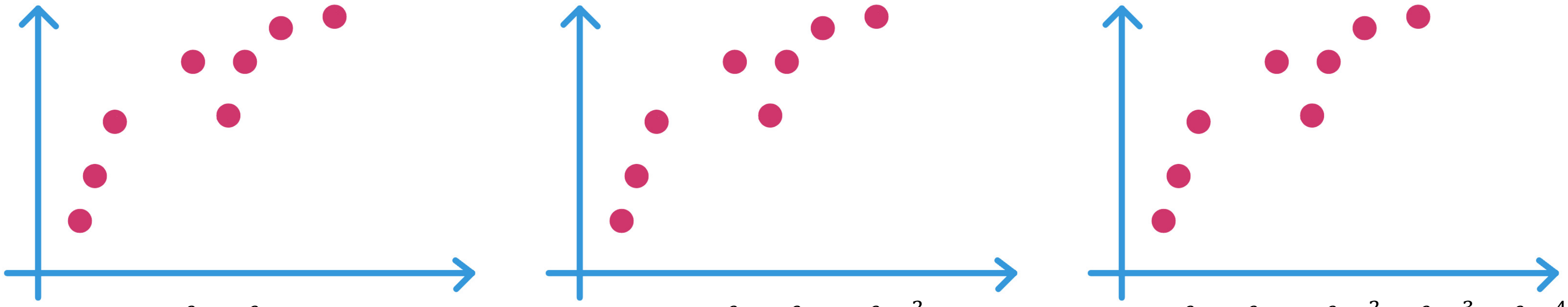
Regularisation

Regularisation

Regularisation is a general method to avoid overfitting by applying additional constraints to the weight vector. A common approach is to make sure the weights are, on average, small in magnitude: this is referred to as *shrinkage*.

Recall the setting for regression in terms of cost minimization.

- Can add penalty terms to a cost function, forcing coefficients to shrink to zero



$y = \theta_0 + \theta_1 x$

$y = \theta_0 + \theta_1 x + \theta_2 x^2$

$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Regularisation

- For example, if MSE is the cost function, the regularized form for data $(x_1, y_1), \dots, (x_m, y_m)$ could be:

$$J(\theta) = \sum_{j=1}^{m} (y_j - h_\theta(x_j))^2 + \lambda \sum_{i=1}^{n} \theta_i^2$$

- Parameter estimation by optimisation attempts to find $\theta_0, \dots, \theta_n$ s.t. $J(\theta)$ is minimum

- Similar to before, this can be solved using Gradient Descent or by taking the derivatives and set them to zero

Regularisation

The multiple least-square regression problem is an optimization problem, as we saw, and can be written as:

$$\theta^* = \arg\min_{\theta}(y - X\theta)^T(y - X\theta)$$

The regularized version of this is then as follows:

$$\theta^* = \arg\min_{\theta}(y - X\theta)^T(y - X\theta) + \lambda||\theta||^2$$

Where $||\theta||^2 = \sum_i \theta_i^2$ is the square norm of the vector θ, or equivalently, the dot product $\theta^T\theta$, and λ is a scalar determining the amount of regularization.

Regularisation

This regularized problem still has a closed-form solution:

$$\theta = (X^T X + \lambda I)^{-1} X^T y$$

where I denotes the identity matrix. Regularisation amounts to adding λ to the diagonal of $X^T X$, a well-known trick to improve the numerical stability of matrix inversion. This form of least-square regression is known as ***Ridge Regression***.

An interesting alternative form of regularization is provided by ***LASSO***, which stand for “Least Absolute Shrinkage and Selection Operator”. It replaces the Ridge regularization term $\sum_i \theta^2$ with the sum of absolute weights $\sum_i |\theta|$. This result in that some weights are shrunk, but others are set to 0, and so the ***LASSO Regression*** *favours sparse solutions*.

Train, Validation & Test Data

- **Train Data:** the data that we use to learn our model and its parameters
- **Validation Data**: The data (unseen by the model) used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters.
- **Test Data:** unseen data by the model that we use to test the model and shows how well our model generalizes. In practice, we don’t know the ground truth (label) for this data

Train, Validation & Test Data

Ideally, we would like, to get the same performance on test set as we get on validation/training set. This is called *Generalization*.

Generalization is the model ability to adapt properly to new, previously unseen data drawn from the same distribution as the one used to create the model.

Model Evaluation

Model Evaluation

The most popular metrics are:

- Root Mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2}$$

- Mean Absolute Error (MAE)

$$MAE = \frac{1}{m}\sum_{j=1}^{m}\left|(y_j - \hat{y}_j)\right|$$

- R-squared ($[-\infty, 1]$)

$$R^2 = 1 - \frac{\sum_{j=1}^{m}(y_j - \hat{y}_j)^2}{\sum_{j=1}^{m}(y_j - \bar{y}\ )^2}$$

- Adjuster R-squared

$$R^2_{adjusted} = 1 - [\frac{(1 - R^2)(m - 1)}{m - n - 1}]$$

Where m is the total number of samples and n is the number of predictors/features.

- R-squared represents the portion of variance in the output that has been explained by the model

Model Evaluation

- the absolute value of RMSE does not actually tell how bad a model is. It can only be used to compare across two models
- Adjusted R-squared easily talks about the quality of the model as well. For example, if a model has adjusted R-squared equal to 0.05 then it is definitely poor.
- However, if you care only about prediction accuracy then RMSE is best. It is computationally simple, easily differentiable and present as default metric for most of the models.

Some further issues in learning linear regression models

Categorical Variables

- “Indicator” variables (or binary variables) are those that take on the values 0 or 1.
- They are used to include the effect of categorical variables, for example, if D is a variable that takes value 1 if a patient takes a drug and 0 if the patient does not. Suppose we want the effect of drug D on blood pressure y, keeping age constant

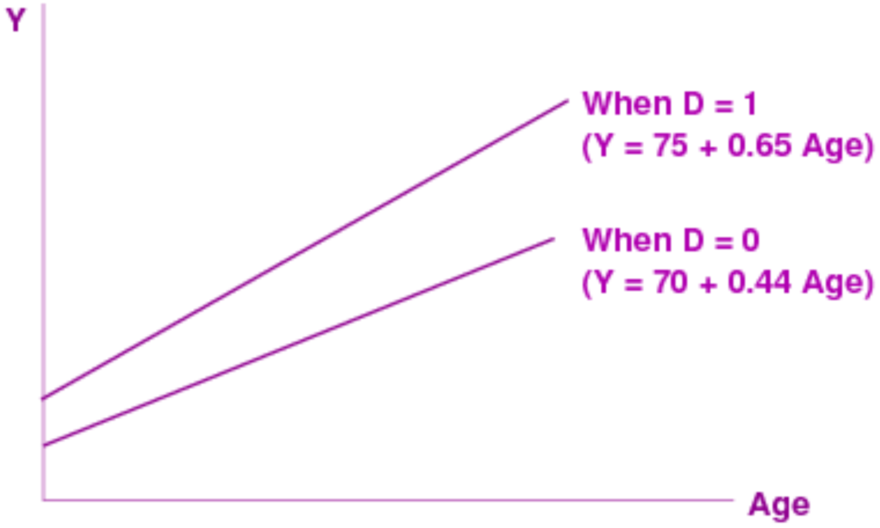
$$y = 70 + 5D + 0.44\ Age$$

- So taking drug D (a unit change in D) makes a difference of 5 units in blood pressure, provided age is held constant.

Categoric Variables

How do we capture any interaction effect between age and drug intake. Introduce a new indicator variable $z = D \times Age$

$$y = 70 + 5D + 0.44\ Age + 0.21\ z$$



Model Selection

Suppose there are a lot of variables/features (x), some of which may be representing products, powers, etc.

- Taking all the features will lead to an overly complex model. There are 3 ways to reduce complexity:
  1. Subset-selection, by search over subset lattice. Each subset results in a new model, and the problem is to select one of the models.
  2. Shrinkage, or regularization of coefficients to zero, by optimization. There is a single model, and unimportant variables have near-zero coefficients.
  3. Dimensionality-reduction, by projecting points into a lower dimensional space (this is different to subset-selection, and we will look at it later)

Model Selection

Subset-selection: we want to find a subset of variables/features which performs well and get rid of redundant features. This is also called stepwise regression.

- Historically, model-selection for regression has been done using “forward-selection”, “backward-elimination”, or “bidirectional” methods
  - These are greedy search techniques that either:
    - (a) start with no variable and at each step add one variable whose addition gives the most improvement to the fit
    - (b) start with all variables and at each step remove one whose loss gives the most insignificant deterioration of the model fit;
    - (c) a combination of the above, testing at each step for variables to be included or excluded.

Model Selection

- This greedy selection done on the basis of calculating the fit quality (often using R-squared, which denotes the proportion of total variation in the dependent variable y that is explained by the model)
- Given a model formed with a subset of variables x, it is possible to compute the observed change in R-squared due to the addition or deletion of some variable x
- This is used to select greedily the next best move in the graph-search

To set other hyper-parameters, such as shrinkage parameter in regularization, we can use grid search

Local (nearest-neighbor) regression

Local Learning

- Relates to the simplest form of learning: rote learning, or memorization
- Training instances are searched for instance that most closely resembles query or test instance
- The instances themselves represent the knowledge
- Called: nearest-neighbor, instance-based, memory-based or case-based learning; all forms of local learning
- The similarity or distance function defines "learning", i.e., how to go beyond simple memorization
- Intuition — predict an instance similarly to examples "close by" — neighbors or exemplars
- A form of lazy learning – don't need to build a model!

Nearest neighbor for numeric prediction

- Store all training examples $(f(x_i), x_i)$

Nearest neighbour

- Given query instance x_q,
- First locate nearest neighbour x_n,
- Then estimate $\hat{y}_q = f(x_n)$

K-nearest neighbour

- Given x_q, take mean of $f$values of k nearest neighbour

$$\hat{y}_q = \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

Distance function

The distance function defines what is “learned”, i.e., predicted

- Most commonly distance function used is Euclidean distance
- If instant x_i is defined with n feature values

$$\langle x_{i1}, \ldots, x_{in} \rangle$$

- The Euclidean distance between two instances x_i, x_j is defined as

$$d(x_i, x_j) = \sqrt{\sum_{k=1}^{n} \left(x_{ik} - x_{jk}\right)^2}$$

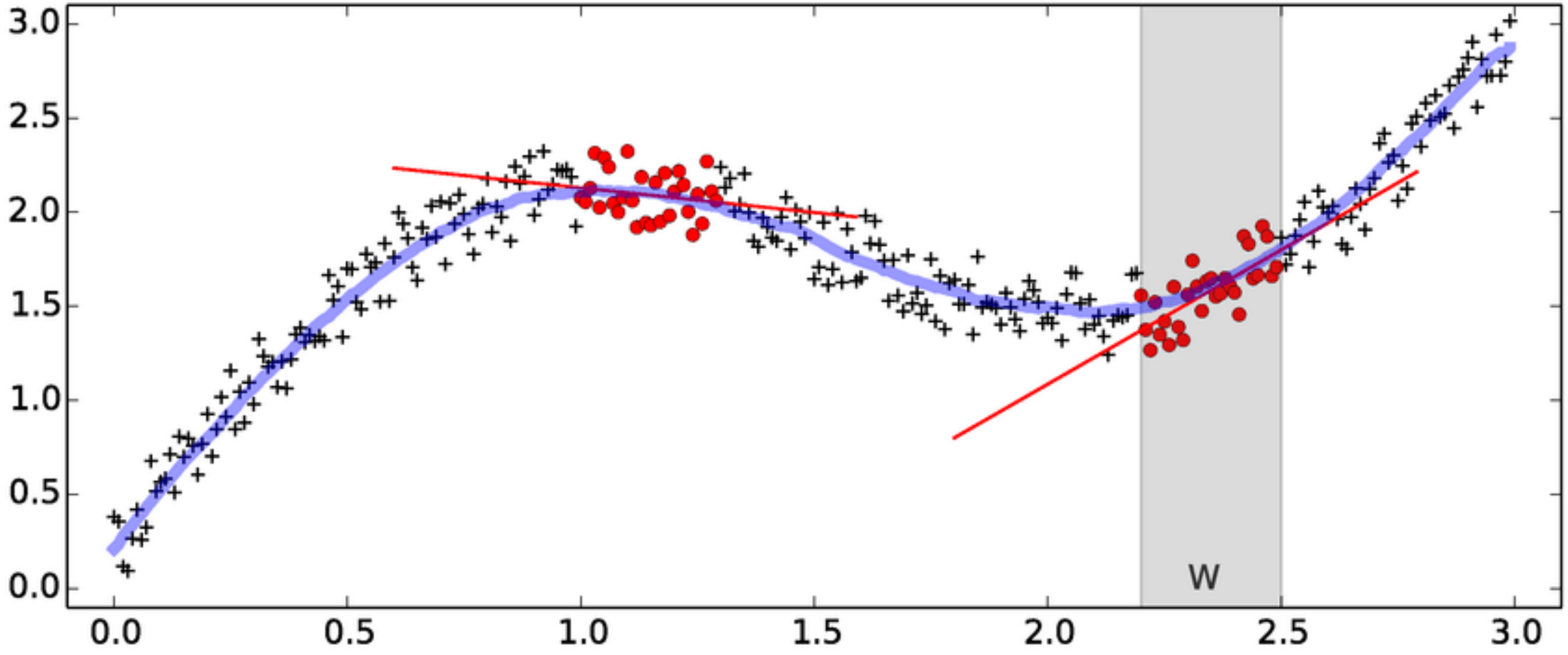
Local regression

Use KNN to form a local approximation to f for each query point x_q using a linear function of the form:

$$\hat{y} = \hat{f}(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$$

- Fit the liner function to k nearest neighbor of the query point
- Linear, quadratic or higher-order polynomial can be used
- Produces “piecewise approximation” so can be useful for non-linear functions, or data with changing distributions

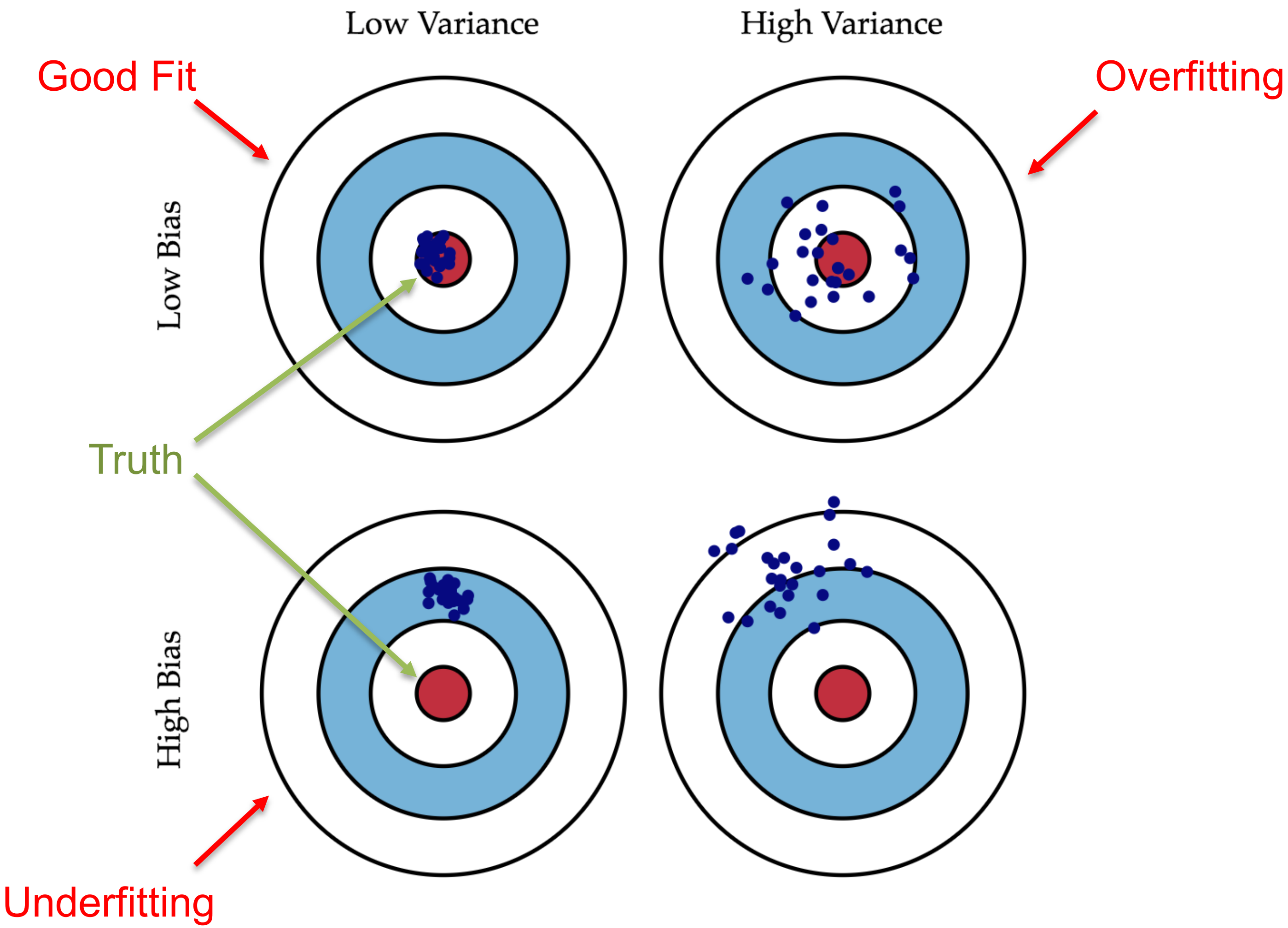
Local regression



[From, Locally-weighted homographies for calibration of imaging systems, by P. Ranganathan & E. Olson]

Bias-Variance Tradeoff

Bias-Variance Tradeoff



Good Fit

Overfitting

Truth

Underfitting

Source: Scott-Fortmann,Understanding Bias-variance tradeoff

Bias-Variance Tradeoff

Bias is the difference between the average prediction of our model and the correct value which we are trying to predict. Model with high bias pays very little attention to the training data and oversimplifies the model. It always leads to high error on training and test data.

Variance is the variability of model prediction for a given data point or a value which tells us spread of our data. Model with high variance pays a lot of attention to training data and does not generalize on the data which it hasn’t seen before. As a result, such models perform very well on training data but has high error rates on test data.

Bias-Variance Tradeoff

- In supervised learning, **underfitting** happens when a model is unable to capture the underlying pattern of the data. These models usually have high bias and low variance.
- In supervised learning, **overfitting** happens when our model captures the noise along with the underlying pattern in data.

Bias-Variance Decomposition

It can be shown that, when we assume $y = f + \varepsilon$ and we estimate f, with $\hat{f}$, then the expectation of error:

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

So., the mean of squared error (MSE) can be written as:

$$MSE = \ Bias^2 + Variance + irreducable\ error$$

- **Irreducible error** or inherent uncertainty is associated with a natural variability in a system (noise). It can not be not reduced since it is due to unknown/unpredictable factors or simply due to chance.

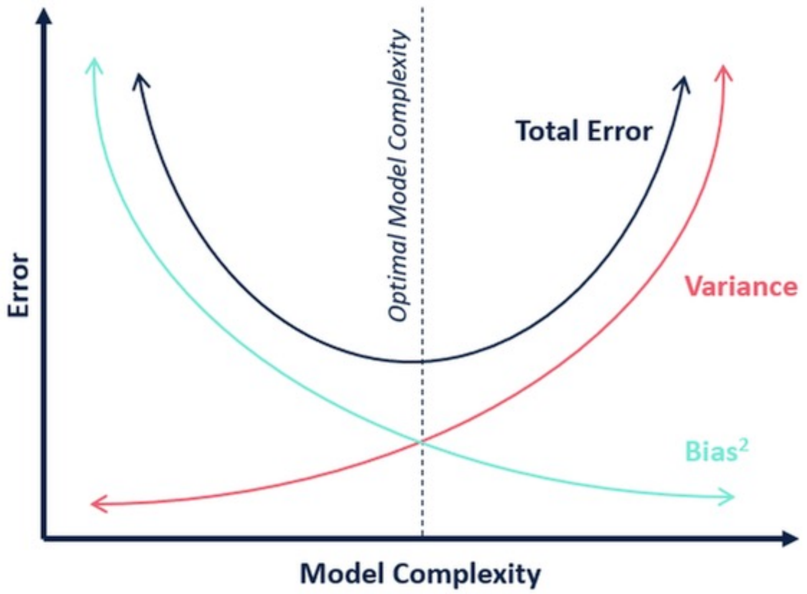
- **Reducible error**, as the name suggests, can be and should be minimized further by adjustments to the model.

Bias-Variance Tradeoff

- When comparing unbiased estimators, we would like to select the one with minimum variance
- In general, we would be comparing estimators that have some bias and some variance
- If our model is too simple and has very few parameters, then it may have high bias and low variance. On the other hand, if our model has large number of parameters then it's going to have high variance and low bias. So, we need to find the right/good balance without overfitting and underfitting the data.
- We need to choose a complexity that makes a good tradeoff between bias and variance.

Bias-Variance Tradeoff

- Often, we want is to find the balance between bias and variance such that we minimize the overall (total) error.
- There is not a quantitative way to find this balanced error point. Instead, you will need to leverage (preferably on an unseen validation data set) and adjust your model's complexity until you find the iteration that minimizes overall error.



Acknowledgements

- Material derived from slides for the book
“Elements of Statistical Learning (2nd Ed.)” by T. Hastie, R. Tibshirani & J. Friedman. Springer (2009) http://statweb.stanford.edu/~tibs/ElemStatLearn/
- Material derived from slides for the book
“Machine Learning: A Probabilistic Perspective” by P. Murphy MIT Press (2012)
http://www.cs.ubc.ca/~murphyk/MLbook
- Material derived from slides for the book “Machine Learning” by P. Flach Cambridge University Press (2012) http://cs.bris.ac.uk/~flach/mlbook
- Material derived from slides for the book
“Bayesian Reasoning and Machine Learning” by D. Barber Cambridge University Press (2012)
http://www.cs.ucl.ac.uk/staff/d.barber/brml
- Material derived from slides for the book “Machine Learning” by T. Mitchell McGraw-Hill (1997)
http://www- 2.cs.cmu.edu/~tom/mlbook.html
- Material derived from slides for the course “Machine Learning” by A. Ng, Stanford University, USA (2015)
- Material derived from slides for the course “Machine Learning” by A. Srinivasan BITS Pilani, Goa, India (2016)