

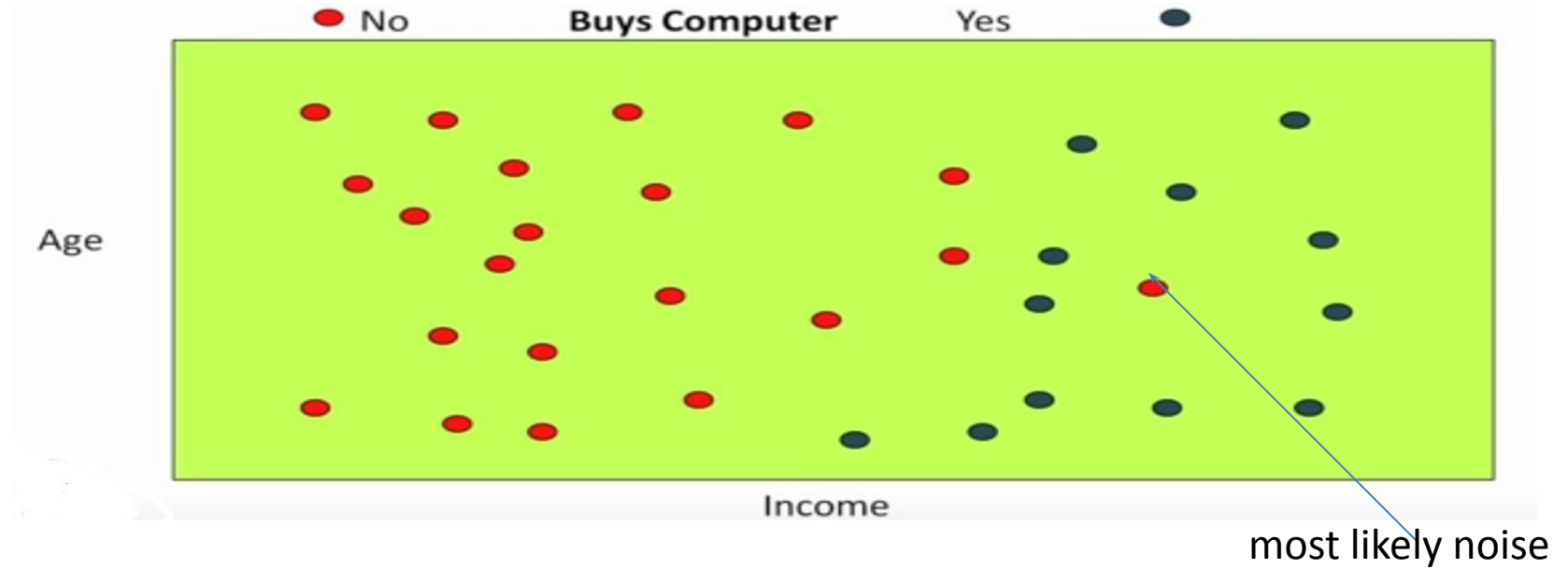
## Unit-II

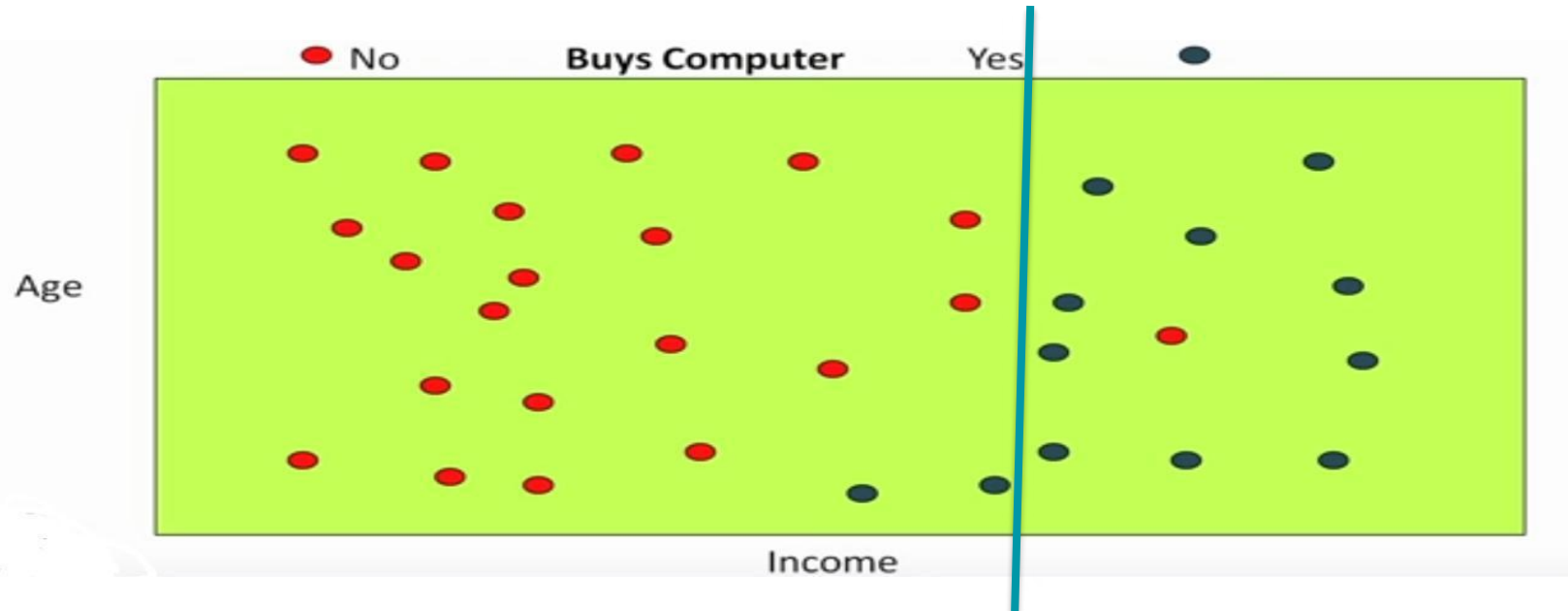
Binary Classification- Assessing Classification performance, Class probability Estimation- Assessing class probability Estimates, Multiclass Classification.

Regression: Assessing performance of Regression- Error measures,  
Overfitting: Catalysts for Overfitting, Case study of Polynomial Regression.

Theory of Generalization: Effective number of hypothesis, Bounding the Growth function, VC Dimensions, Regularization theory.

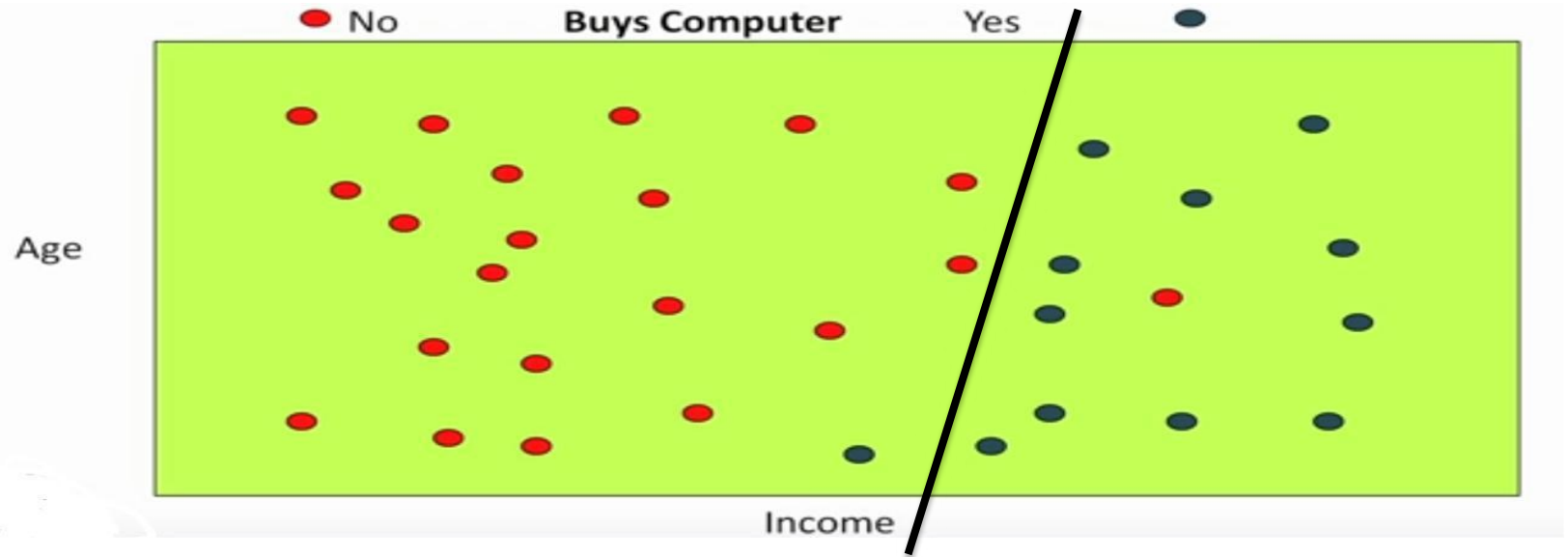
# Experience (training data)

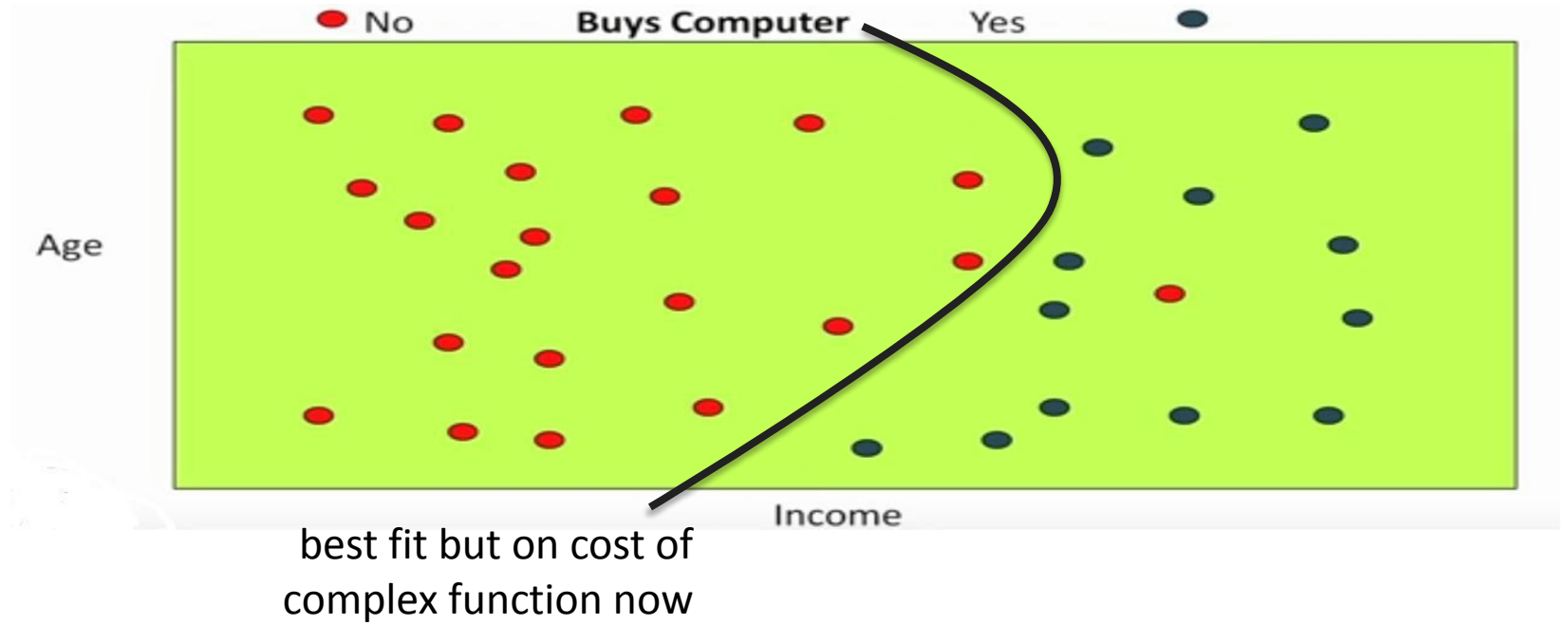




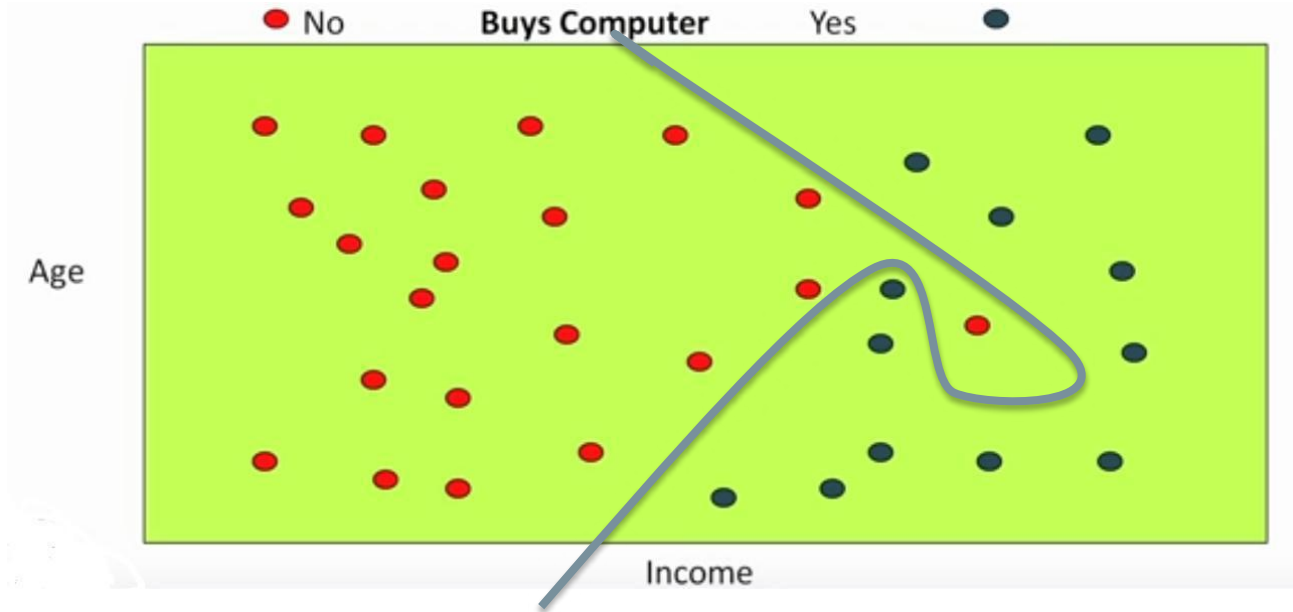
dependant on just income

dependant on income & age





too complex

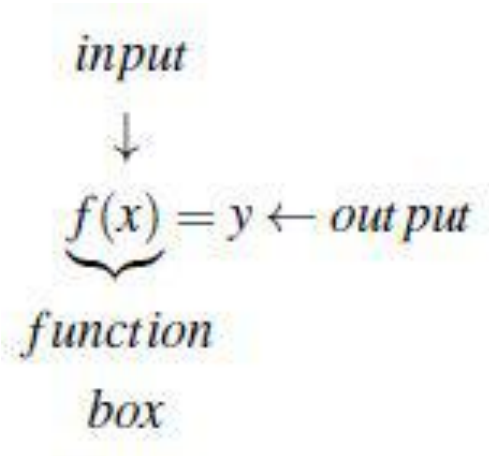


# FUNCTIONS AS MODELS

# FUNCTION

A function is a set of ordered pairs in which the first coordinate, usually  $x$ , matches with exactly one second coordinate,  $y$ . Equations that follow this definition can be written in function notation. The  $y$  coordinate represents the dependent variable, meaning the values of this variable depend upon what is substituted for the other variable.

A function can be expressed as an equation, as shown below. In the equation,  $f$  represents the function name and  $(x)$  represents the variable. In this case the parentheses do not mean multiplication; rather, they separate the function name from the independent variable.

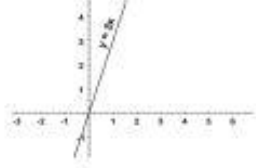




# Representation

Functions may be presented in many ways. Some of the most common ways to represent functions include:

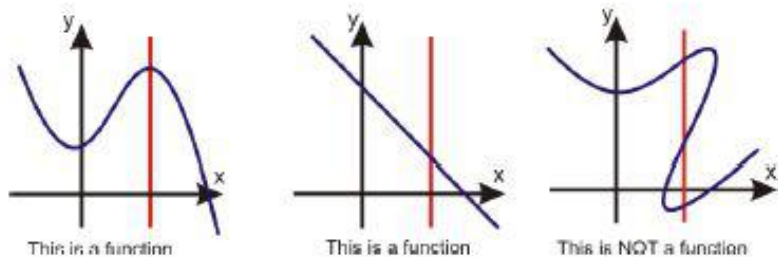
- a graph
- ordered pairs
- an equation
- a table of values
- an arrow or mapping diagram

Representation	Example
Set of ordered pairs	(1,3), (2,6), (3,9), (4,12) (a subset of the ordered pairs for this function)
Equation	$y = 3x$
Graph	

The figure above actually shows the same function depicted in three different ways! In the first representation, we are given a set of ordered pairs. To verify that this is a function, we must ensure that each x-value is associated with a single y-value. In this example, the first number in each pair (the x-value) is different, so we can be certain that there are no cases where a particular x is associated with more than one y.

In the second representation, the equation of a line, it is apparent that any number put in place of x will result in a different y, since the x number is simply being multiplied by 3.

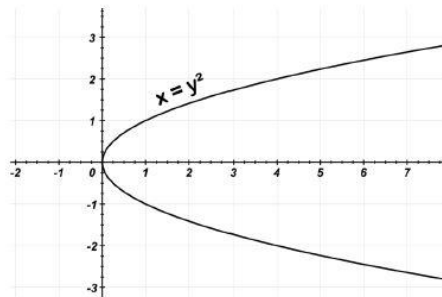
The third representation above is a graph. A good way to determine whether a relation is a function when looking at a graph is by doing a "vertical line test". If a vertical line can be drawn anywhere on the graph such that the line crosses the relation in two places, then the relation is not a function. If all possible vertical lines will only cross the relation in one place, then the relation is a function.



The vertical line test works because if a vertical line crosses a relation in more than one place it means that there must be two  $y$  values corresponding to one  $x$  value in that relation.

The graph above of  $y = 3x$  shows it is a function because any vertical line that is drawn only crosses the relation in one place.

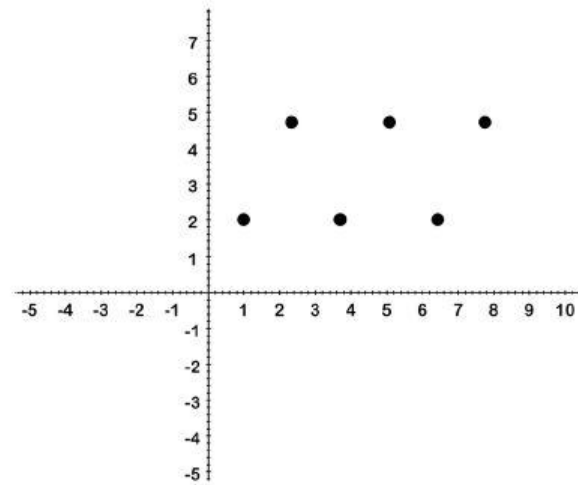
Conversely, the graph below of  $x = y^2$  shows it is not a function because a vertical line can be drawn that crosses the relation in two places.



Determine if each relation is a function:

a.  $(2, 4)$ ,  $(3, 9)$ ,  $(5, 11)$ ,  $(5, 12)$

b. see graph

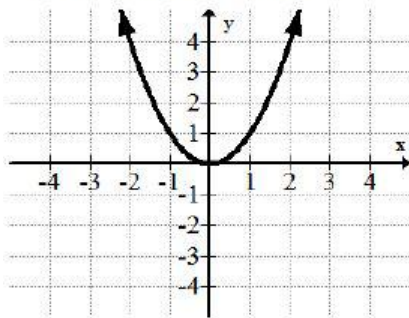


### *Solutions*

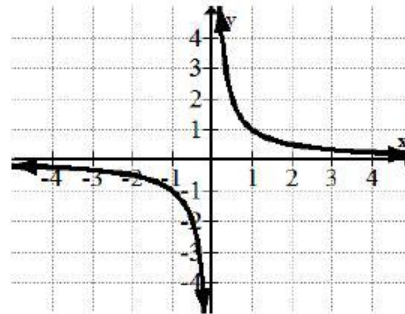
a. This relation is not a function because 5 is paired with 11 and with 12.

b. This relation is a function because every  $x$  is paired with only one  $y$ . A vertical line through the graph will always only encounter a single point

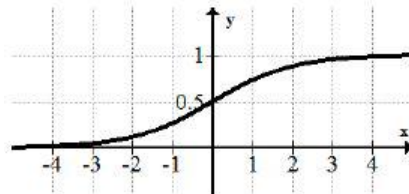
The Squaring Function:  $f(x) = x^2$



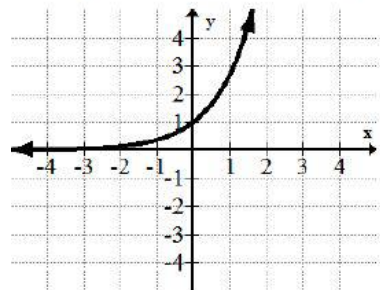
The Reciprocal Function:  $f(x) = x^{-1} = \frac{1}{x}$



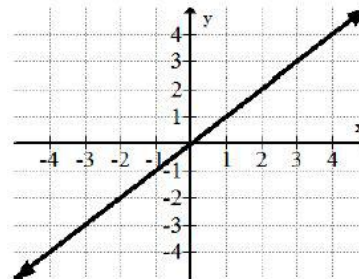
The Logistic Function:  $f(t) = \frac{C}{1+ab^{-t}} = \frac{C}{1+ae^{-kt}}$



The Exponential Function Family:  $f(x) = e^x$



The Identity (Linear) Function:  $f(x) = x$



# REGRESSION

- **Numerical prediction** is similar to **classification**
  - construct a model
  - use model to predict continuous or ordered value for a given input
- **Numeric prediction vs. classification**
  - Classification refers to predict categorical class label
  - Numeric prediction models continuous-valued functions

- **Regression analysis** is the major method for numeric prediction
- **Regression analysis** model the relationship between
  - one or more **independent** or **predictor variables** and
  - a **dependent** or **response** variable
- **Regression analysis** is a good choice when all of the **predictor variables** are **continuous** valued as well.

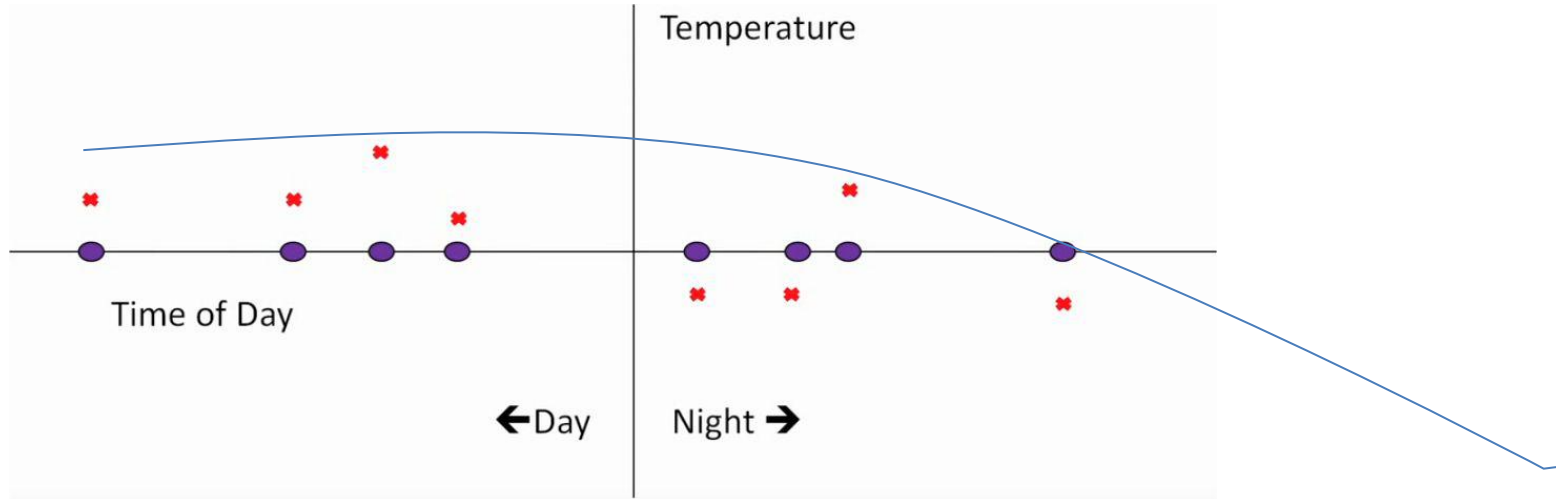


- In the context of data mining
  - The **predictor variables** are the **attributes** of interest describing the instance that are known.
  - The response variable is what we want to predict
- Some classification techniques can be adapted for prediction, e.g.
  - Backpropagation
  - k-nearest-neighbor classifiers
  - Support vector machines

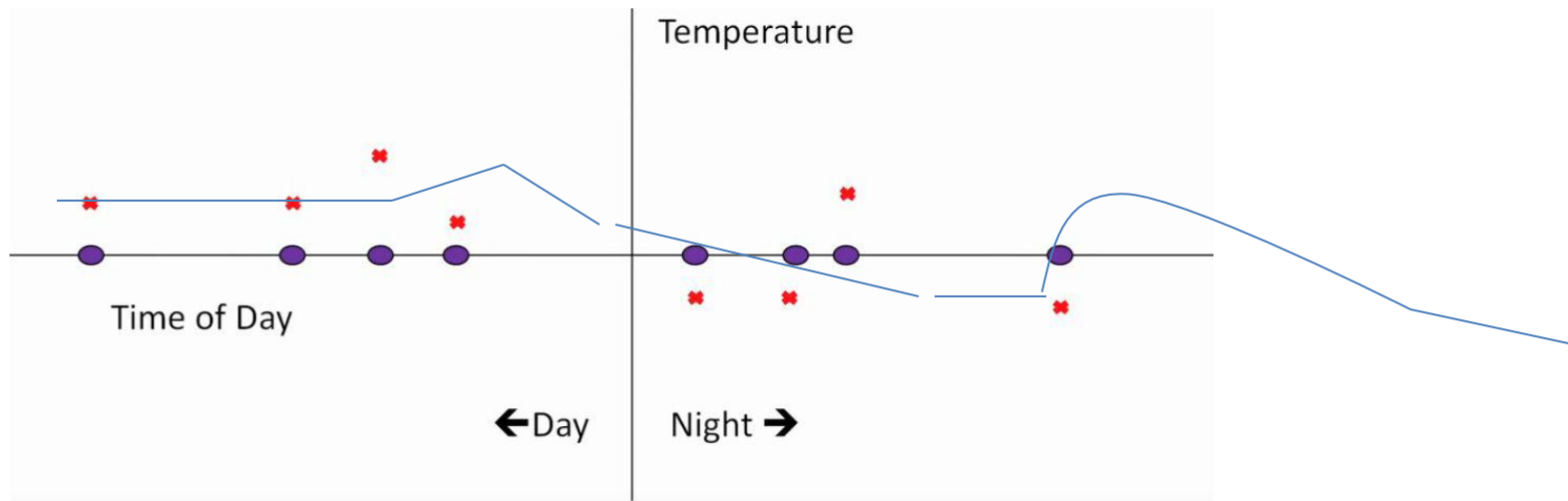
- **Regression analysis methods:**

- Linear regression
  - ◆ Straight-line linear regression
  - ◆ Multiple linear regression
- Non-linear regression
- Generalized linear model
  - ◆ Poisson regression
  - ◆ Logistic regression
- Log-linear models
- Regression trees and Model trees

# Curve



# Overfitting



# Applications

Time series prediction

Rainfall in a certain region

Spend on voice calls

Classification ( by encoding class labels as values)

Data Reduction (coefficients of curve may be given to get the data trend)

Trend Analysis (Run time trends)

Linear or exponential

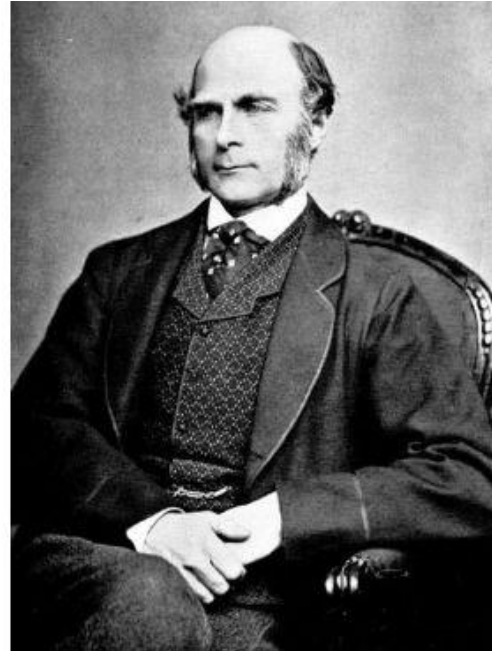
Risk factor analysis

Factors contributing most to output

# LINEAR REGRESSION

# History

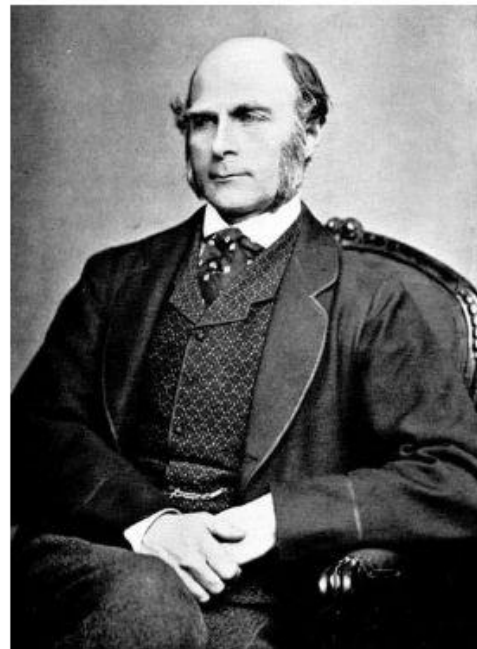
This all started in the 1800s with a guy named **Francis Galton**. Galton was studying the relationship between parents and their children. In particular, he investigated the relationship between the heights of fathers and their sons.



# History

What he discovered was that a man's son tended to be roughly as tall as his father.

However Galton's breakthrough was that the son's height **tended to be closer to the overall average** height of all people.

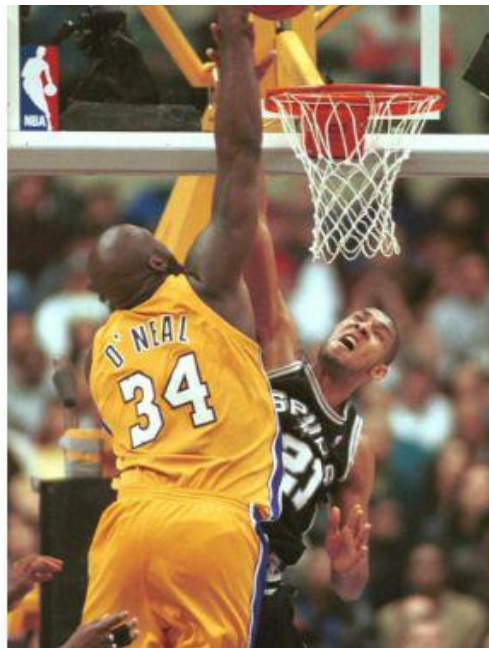




# Example

Let's take **Shaquille O'Neal** as an example. Shaq is really tall: 7ft 1in (2.2 meters).

If Shaq has a son, chances are he'll be pretty tall too. However, Shaq is such an anomaly that there is also a very good chance that his son will be **not be as tall as Shaq**.

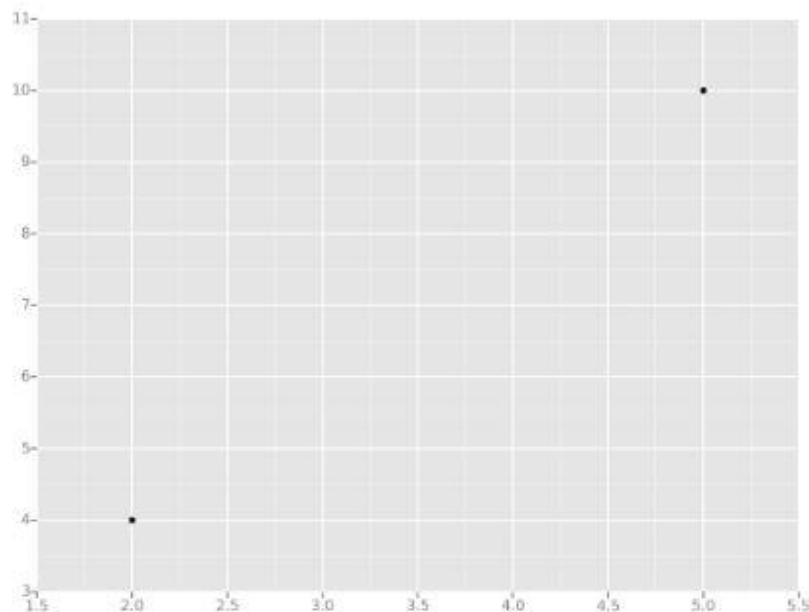


Turns out this is the case:  
Shaq's son is pretty tall (6 ft 7 in), but not nearly as tall as his dad.

Galton called this phenomenon **regression**, as in "A father's son's height tends to regress (or drift towards) the mean (average) height."

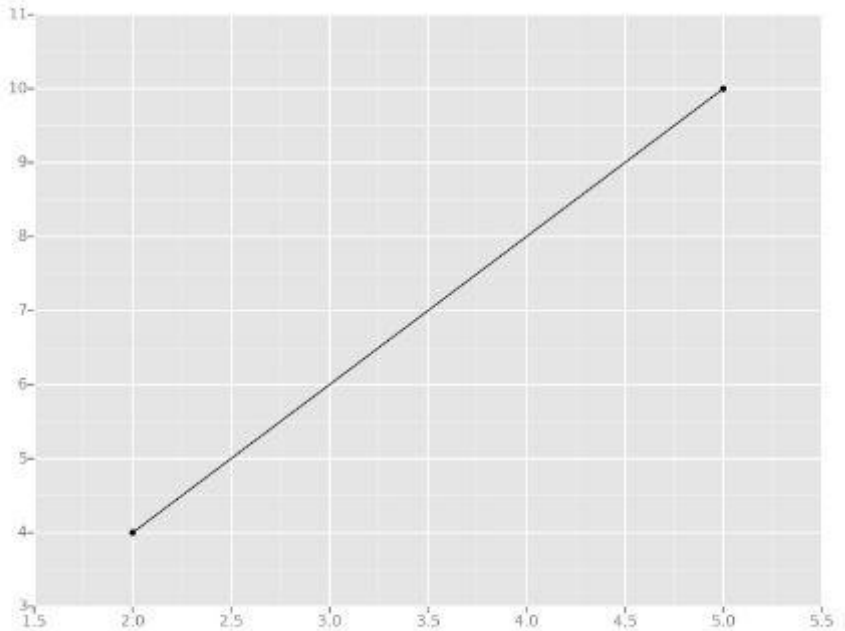


Let's take the simplest possible example: calculating a regression with only 2 data points.



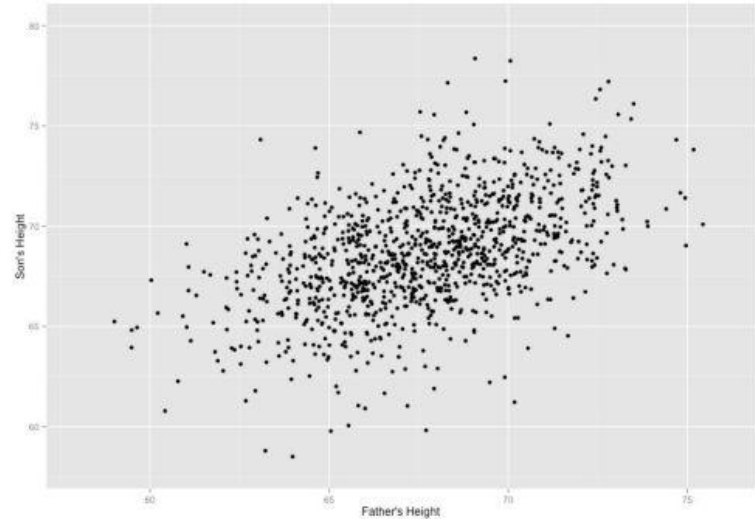
All we're trying to do when we calculate our regression line is draw a line that's as close to every dot as possible.

For classic linear regression, or "Least Squares Method", you only measure the closeness in the "up and down" direction



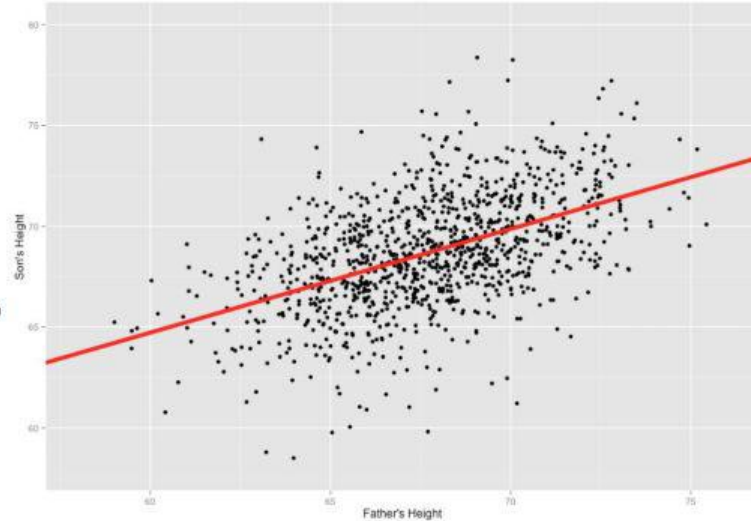
Now wouldn't it be great if we could apply this same concept to a graph with more than just two data points?

By doing this, we could take multiple men and their son's heights and do things like tell a man how tall we expect his son to be...before he even has a son!



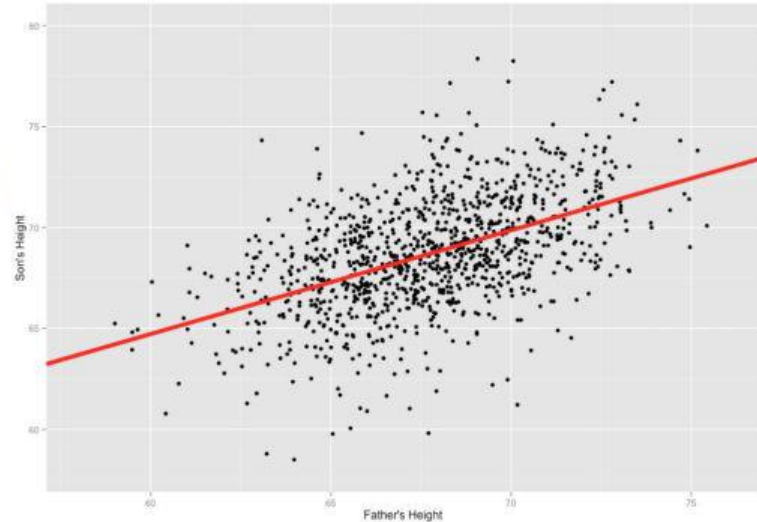
Our goal with linear regression is to **minimize the vertical distance** between all the data points and our line.

So in determining the **best line**, we are attempting to minimize the distance between **all** the points and their distance to our line.



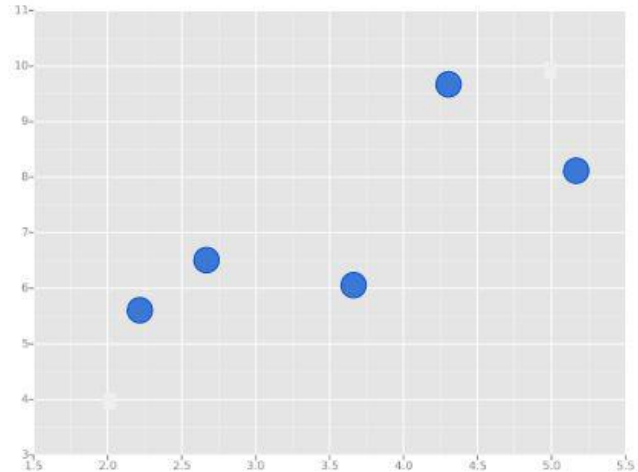


There are lots of different ways to minimize this, (sum of squared errors, sum of absolute errors, etc), but all these methods have a general goal of minimizing this distance.



For example, one of the most popular methods is the least squares method.

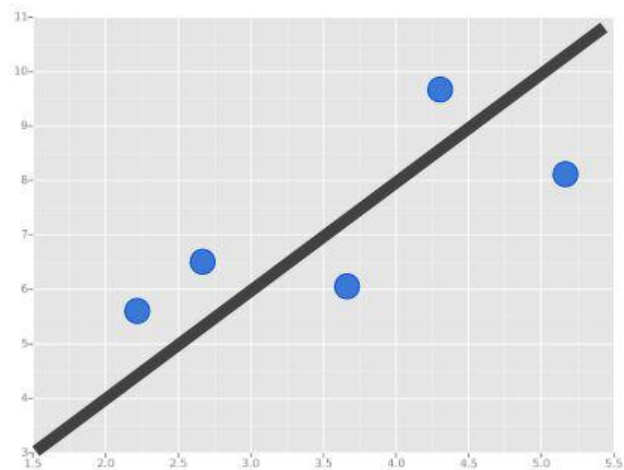
Here we have blue data points along an x and y axis.





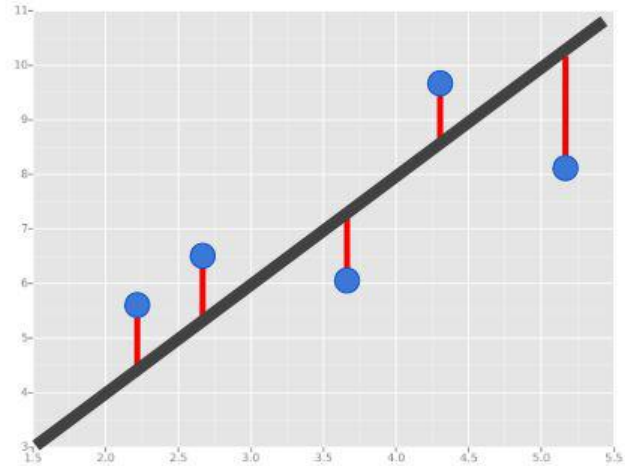
Now we want to fit a linear regression line.

The question is, how do we decide which line is the best fitting one?

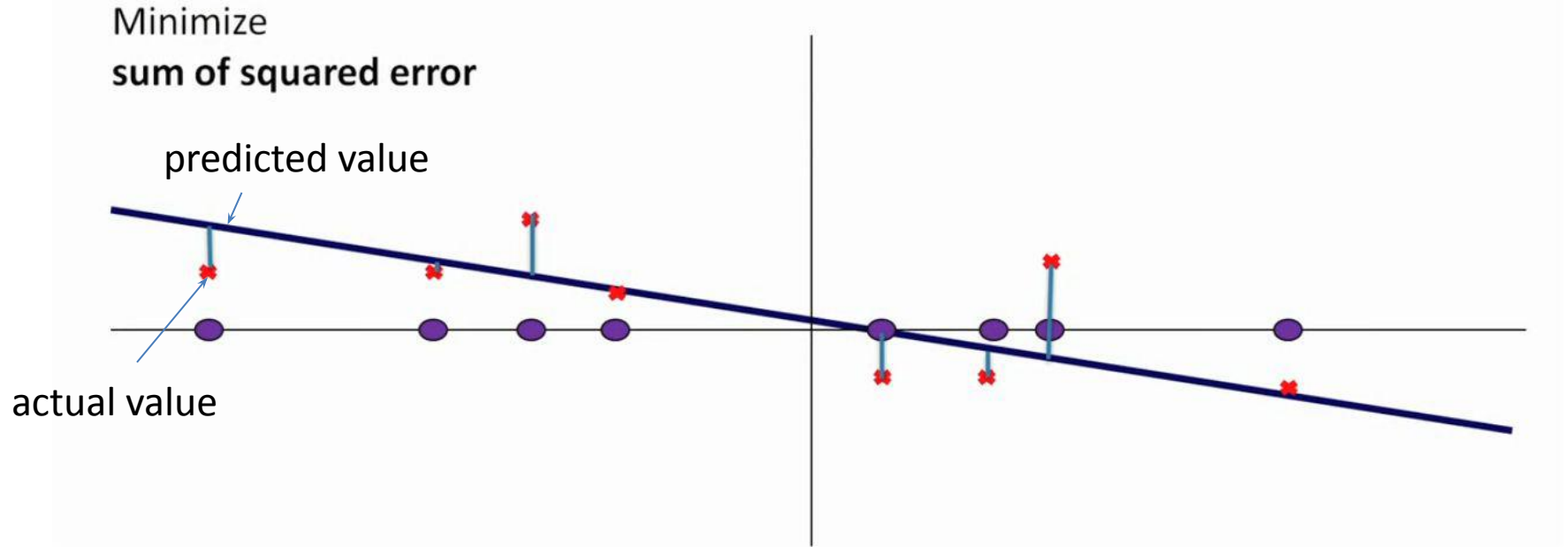


We'll use the Least Squares Method, which is fitted by minimizing the ***sum of squares of the residuals***.

The residuals for an observation is the difference between the observation (the y-value) and the fitted line.



# Linear Regression



# Linear Regression

- **Straight-line linear regression:**

- involves a response variable  $y$  and a single predictor variable  $x$

$$y = w_0 + w_1 x$$

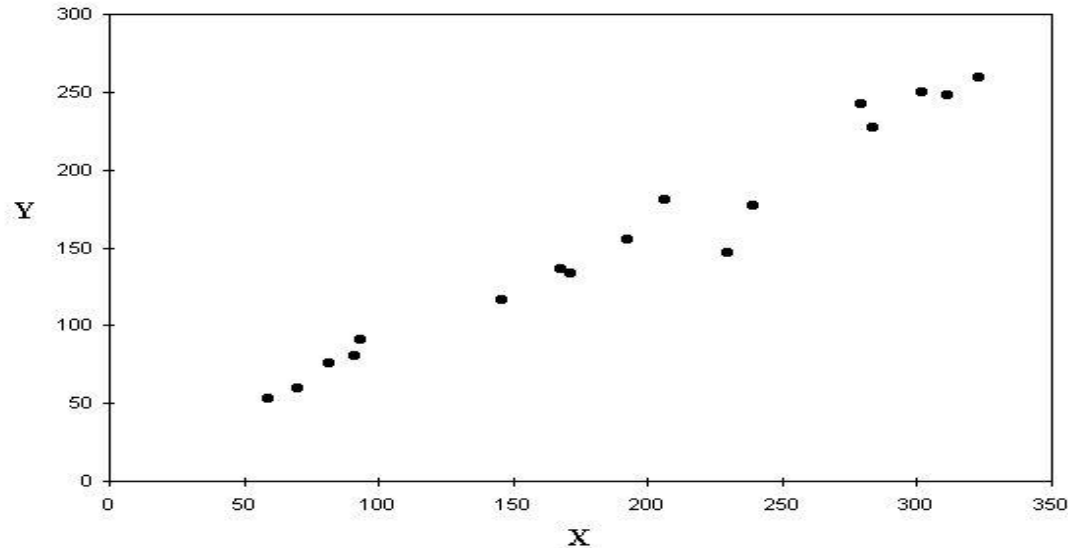
- $w_0$  :  $y$ -intercept
- $w_1$  : slope
- $w_0$  &  $w_1$  are **regression coefficients**

# Least Square Error

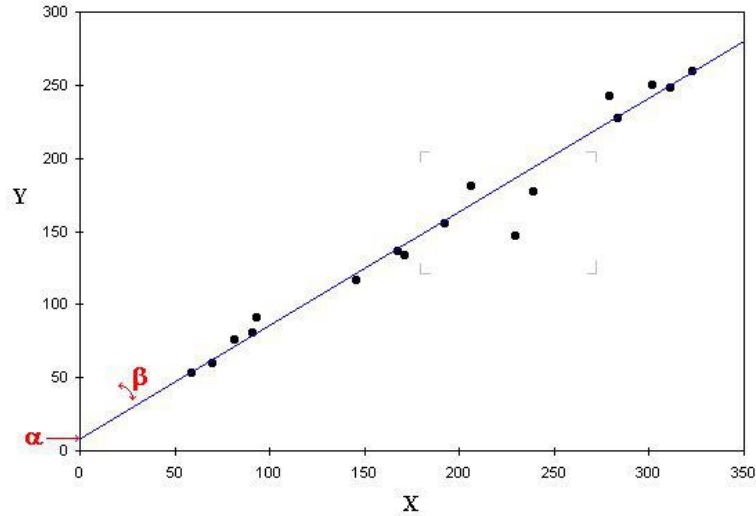
- **Method of least squares**: estimates the best-fitting straight line as the one that minimizes the error between the actual data and the estimate of the line.

$$w_1 = \frac{\sum_{i=1}^{|D|} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{|D|} (x_i - \bar{x})^2} \quad w_0 = \bar{y} - w_1 \bar{x}$$

- $D$ : a training set
- $x$ : values of predictor variable
- $y$ : values of response variable
- $|D|$ : data points of the form  $(x_1, y_1), (x_2, y_2), \dots, (x_{|D|}, y_{|D|})$ .
- $\bar{x}$ : the mean value of  $x_1, x_2, \dots, x_{|D|}$
- $\bar{y}$ : the mean value of  $y_1, y_2, \dots, y_{|D|}$

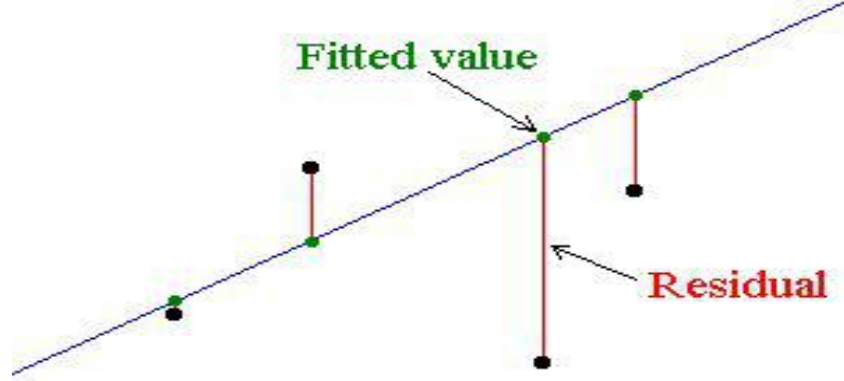


Let's begin by looking at the simplest case, where there are two variables, one explanatory (X) and one response variable (Y), *ie.* change in X causes a change in Y. It is always worth viewing your data (if possible) before performing regressions to get an idea as to the type of relationship (*eg.* whether it is best described by a straight line or curve).



By looking at this scatter plot, it can be seen that variables X and Y have a close relationship that may be reasonably represented by a straight line. This would be represented mathematically as  $Y = a + bX + e$

where  $a$  describes where the line crosses the y-axis,  $b$  describes the slope of the line, and  $e$  is an error term that describes the variation of the real data above and below the line. Simple linear regression attempts to find a straight line that best 'fits' the data, where the variation of the real data above and below the line is minimised.



The fitted line has  $a=7.76$  and  $b=0.769$  and now that we know the equation, we can plot the line onto the data (e is not needed to plot the line); see Fig 2.1 below. This is the mathematical model describing the functional response of Y to X.

The R-squared and adjusted R-squared values are estimates of the 'goodness of fit' of the line. They represent the % variation of the data explained by the fitted line; the closer the points to the line, the better the fit. Adjusted R-squared is not sensitive to the number of points within the data. R-squared is derived from

$$\text{R-squared} = 100 * \text{SS}(\text{regression}) / \text{SS}(\text{total})$$



# SS

For linear regression with one explanatory variable like this analysis, R-squared is the same as the square of  $r$ , the correlation coefficient.

The sum of squares (SS) represents variation from several sources.

**SS(regression)** describes the variation within the fitted values of  $Y$ , and is the sum of the squared difference between each fitted value of  $Y$  and the mean of  $Y$ . The squares are taken to 'remove' the sign (+ or -) from the residual values to make the calculation easier.

**SS(error)** describes the variation of observed  $Y$  from estimated (fitted)  $Y$ . It is derived from the cumulative addition of the square of each residual, where a residual is the distance of a data point above or below the fitted

**SS(total)** describes the variation within the values of  $Y$ , and is the sum of the squared difference between each value of  $Y$  and the mean of  $Y$ .

Now, interestingly, the "coefficient of determination" is also called the "proportion of variance accounted for" because that's what it IS.

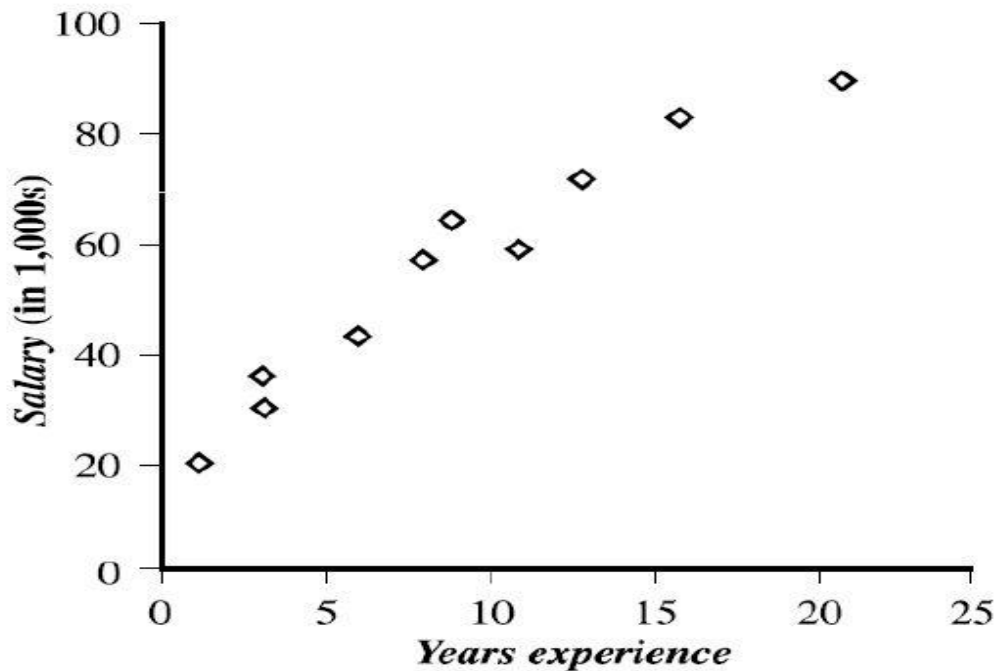
It's the amount of error that we can account for in the variable of interest - the dependent variable - by knowing the model of the relationship with it and our independent variable.

The table shows a set of paired data where  $x$  is the number of years of work experience of a college graduate and  $y$  is the corresponding salary of the graduate.

$x$ years experience	$y$ salary (in \$1000s)
3	30
8	57
9	64
13	72
3	36
6	43
11	59
21	90
1	20
16	83

The 2-D data can be graphed on a **scatter plot**.

The plot suggests a linear relationship between the two variables,  $x$  and  $y$ .



- Given the above data, we compute

$$\bar{x} = 9.1 \text{ and } \bar{y} = 55.4$$

- we get

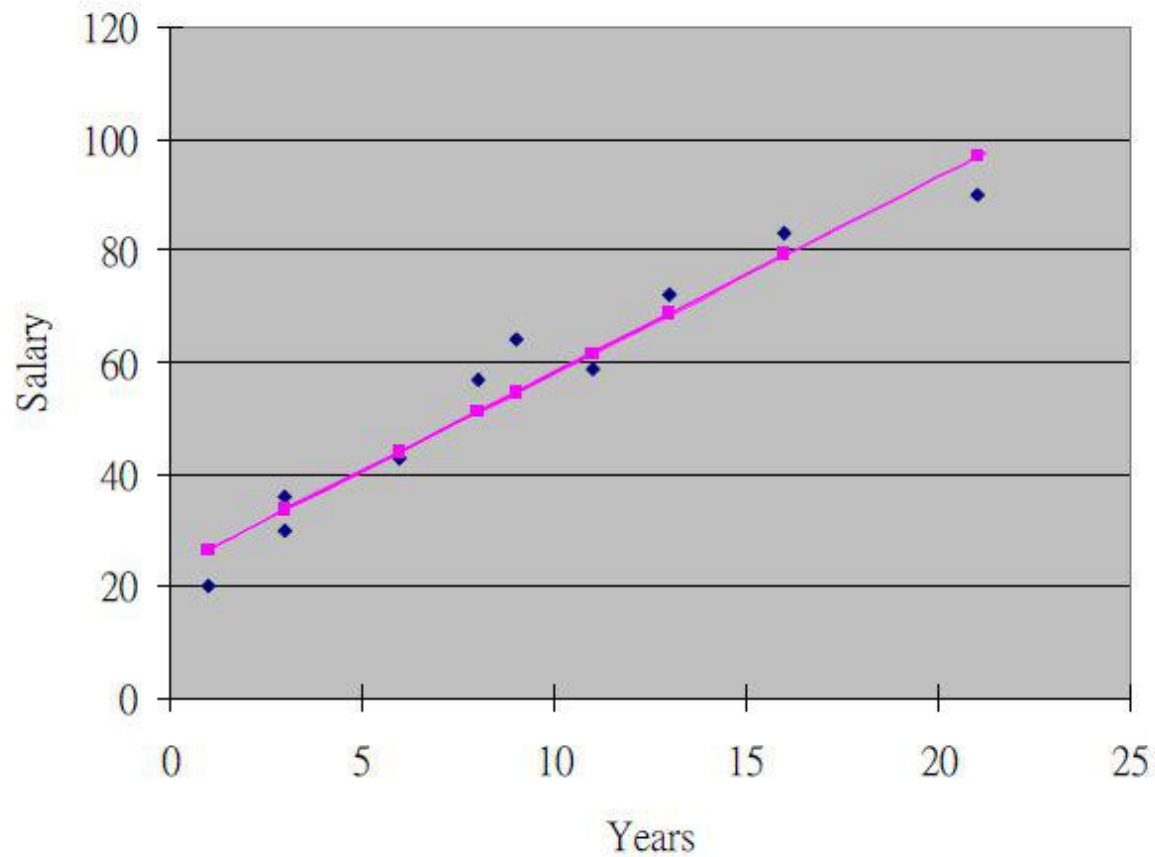
$$w_1 = \frac{(3 - 9.1)(30 - 55.4) + (8 - 9.1)(57 - 55.4) + \dots + (16 - 9.1)(83 - 55.4)}{(3 - 9.1)^2 + (8 - 9.1)^2 + \dots + (16 - 9.1)^2} = 3.5$$

$$w_0 = 55.4 - (3.5)(9.1) = 23.6$$

- The equation of the least squares line is estimated by

$$y = 23.6 + 3.5x$$

Linear Regression:  $Y=3.5*X+23.2$



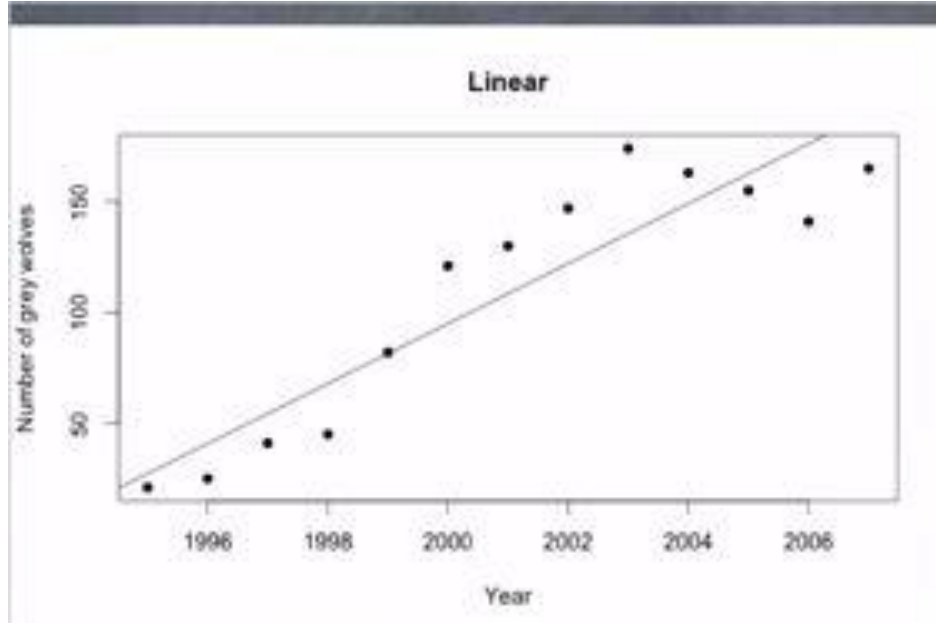
$$f(x) = 46.91 + (-0.51)x$$

Free Lunches	Helmets
20	36.68
21	36.17

↘  
-0.51  
↙

A stable feature of the linear model:

a constant rate of change over equally spaced distances of the input variable.





# Model

Here's some data based on **Yellowstone's re-introduction of the grey wolf, starting in 1995.**

We can see that as time goes by, the number of wolves increases.

Let's fit our linear model to this data.

We see our results of running the linear model: An R-squared of 0.840, an Intercept value of -26,982, and a slope of 13.54.

## How to make sense?

Our slope is the rate of change in our predicted outcome variable we expect to see with one whole unit increase in our x-axis variable.

Our intercept is the predicted value of our outcome variable **when our x-axis variable is ZERO.**

# Solution

Well, the easiest way, is to make the x-axis - the values of our independent variable - more usable to us.

So instead of using just years, let's start our years, for our example, at the time that the wolves were reintroduced back into Yellowstone - at 1995.

**So we'll make a new x variable, that will effectively make 1995 OUR year zero - making the x-axis the years since reintroduction, or years since 1995.**

Now, if we re-run the linear model with the new independent variable

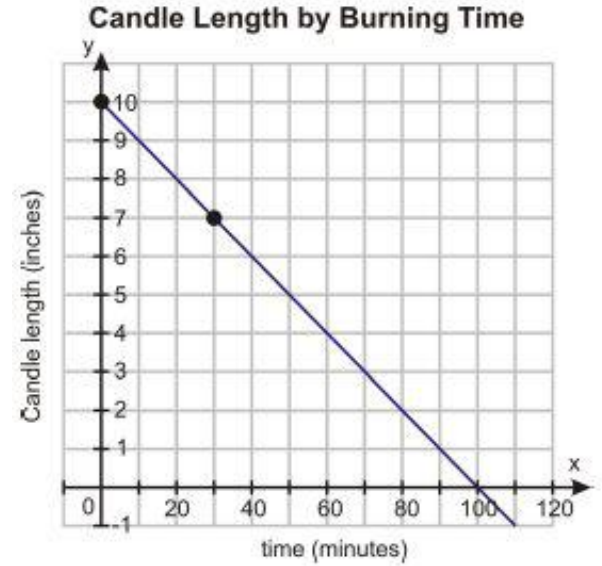
# LINEAR REGRESSION MODELS

The slope of a function that describes real, measurable quantities is often called a rate of change. In that case, the slope refers to a change in one quantity ( $y$ ) per unit change in another quantity ( $x$ ).

A candle has a starting length of 10 inches. Thirty minutes after lighting it, the length is 7 inches. Determine the rate of change in length of the candle as it burns. Determine how long the candle takes to completely burn to nothing.

We have two points to start with. We know that at the moment the candle is lit (time = 0) the length of the candle is 10 inches. After thirty minutes (time = 30) the length is 7 inches. Since the candle length is a function of time we will plot time on the horizontal axis, and candle length on the vertical axis.

The rate of change of the candle is simply the slope. Since we have our two points  $(x_1, y_1) = (0, 10)$  and  $(x_2, y_2) = (30, 7)$  we can move straight to the formula.



$$\text{Rate of change} = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} = \frac{(7 \text{ inches}) - (10 \text{ inches})}{(30 \text{ minutes}) - (0 \text{ minutes})} = \frac{-3 \text{ inches}}{30 \text{ minutes}} = -0.1 \text{ inches per minute}$$

$$\text{rate} = \frac{-0.1 \text{ inches}}{1 \text{ minute}} \cdot \frac{60 \text{ minutes}}{1 \text{ hour}} = \frac{-6 \text{ inches}}{1 \text{ hour}} = -6 \text{ inches per hour}$$

To find the point when the candle burns to nothing, or reaches zero length, we can read off the graph (100 minutes).

We can use the rate equation to verify this algebraically.

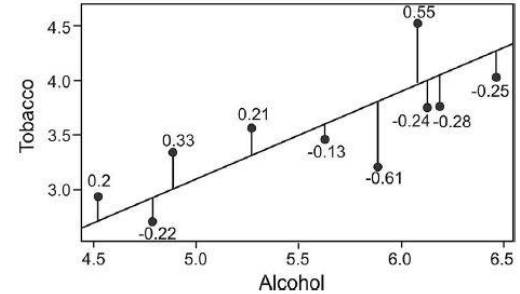
Rate \* time = Length burned

$$0.1 * 100 = 10$$

Since the candle length was originally 10 inches this confirms that 100 minutes is the correct amount of time.

# Generating and Graphing the Regression Line

Earlier we learned that correlation could be used to assess the strength and direction of a linear relationship between two variables. We illustrated the concept of correlation through scatterplot graphs.



We saw that when variables were correlated, the points on a scatterplot graph tended to follow a straight line. If we could draw this straight line, it would, in theory, represent the change in one variable associated with the change in the other. This line is called the least squares line, or the linear regression line, or the line of best fit.

Linear regression involves using data to calculate a line that best fits that data, and then using that line to predict scores on one variable from another.

Prediction is simply the process of estimating scores of the outcome (or dependent) variable based on the scores of the predictor (or independent) variable.



To generate the regression line, we look for a line of best fit. There are many ways one could define this best fit.

Statisticians define the best-fit line to be the one that minimizes the sum of the squared distances from the observed data to the line. This method of fitting the data line so that there is minimal difference between the observations and the line is called the method of least squares.

Our goal in the method of least squares is to fit the regression line to the data by having the smallest sum of squared distances possible from each of the data points to the line. In the example shown, you can see the calculated distances, or residual values, from each of the observations to the regression line. The smaller the residuals, the better the fit.

As you can see, the regression line is a straight line that expresses the relationship between two variables. Since the regression line is used to predict the value of Y for any given value of X, all predicted values will be located on the regression line itself.

When predicting one score by using another, we use an equation such as the following, which is equivalent to the slope-intercept form of the equation for a straight line:

$$\hat{Y} = bX + a$$

$\hat{Y}$  (pronounced Y-hat) is the score that we are trying to predict.

b is the slope of the line; it is also called the regression coefficient.

a is the y-intercept; it is also called the regression constant. It is the value of Y when the value of X is 0.

# Generating a Linear Model from Raw Data

To calculate a regression line from our data, we need to find the values for  $b$  and  $a$ . To calculate the regression coefficient  $b$  (or slope), we can use the following formulas:

$$b = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2}$$

or

$$b = (r) \frac{s_Y}{s_X}$$

where:

$r$  is the correlation between the variables  $X$  and  $Y$ .

$s_Y$  is the standard deviation of the  $Y$  scores.

$s_X$  is the standard deviation of the  $X$  scores.

To calculate the regression constant  $a$  (or y-intercept), we use the following formula:

$$a = \frac{\sum y - b \sum x}{n}$$

or

$$a = \bar{y} - b\bar{x}$$

Find the least squares line (also known as the linear regression line or the line of best fit) for the example measuring the verbal SAT scores and GPAs of students that was used in the previous section.

Student	SAT Score ( $X$ )	GPA ( $Y$ )	$xy$	$x^2$	$y^2$
1	595	3.4	2023	354025	11.56
2	520	3.2	1664	270400	10.24
3	715	3.9	2789	511225	15.21
4	405	2.3	932	164025	5.29
5	680	3.9	2652	462400	15.21
6	490	2.5	1225	240100	6.25
7	565	3.5	1978	319225	12.25
Sum	3970	22.7	13262	2321400	76.01

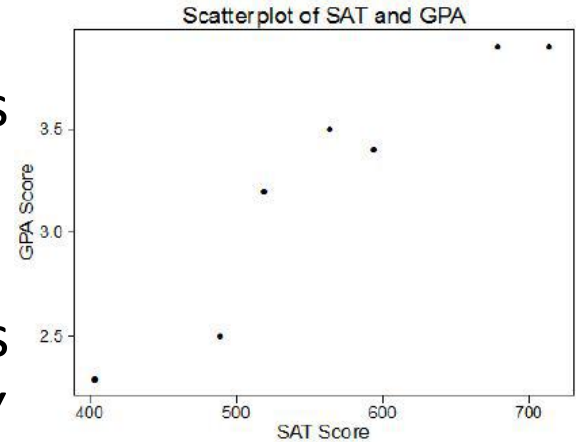
$$b = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - (\sum x)^2} = \frac{(7)(13,262) - (3,970)(22.7)}{(7)(2,321,400) - 3,970^2} = \frac{2715}{488900} \approx 0.0056$$

$$a = \frac{\sum y - b \sum x}{n} \approx 0.094$$

Now that we have the equation of this line, it is easy to plot on a scatterplot. To plot this line, we simply substitute two values of  $X$  and calculate the corresponding  $Y$  values to get two pairs of coordinates.

Let's say that we wanted to plot this example on a scatterplot.

We would choose two hypothetical values for  $X$  (say, 400 and 500) and then solve for  $Y$  in order to identify the coordinates (400, 2.334) and (500, 2.89). From these pairs of coordinates, we can draw the regression line on the scatterplot.



One of the uses of a regression line is to predict values. After calculating this line, we are able to predict values by simply substituting a value of a predictor variable,  $X$ , into the regression equation and solving the equation for the outcome variable,  $Y$ . In our example above, we can predict the students' GPA's from their SAT scores by plugging in the desired values into our regression equation,

$$\hat{Y} = 0.0056X + 0.094$$

For example, say that we wanted to predict the GPA for two students, one who had an SAT score of 500 and the other who had an SAT score of 600. To predict the GPA scores for these two students, we would simply plug the two values of the predictor variable into the equation and solve for  $Y$

Student	SAT Score ( $X$ )	GPA ( $Y$ )	Predicted GPA ( $\hat{Y}$ )
1	595	3.4	3.4
2	520	3.2	3.0
3	715	3.9	4.1
4	405	2.3	2.3
5	680	3.9	3.9
6	490	2.5	2.8
7	565	3.5	3.2
Hypothetical	600		3.4
Hypothetical	500		2.9

As you can see, we are able to predict the value for  $Y$  for any value of  $X$  within a specified range.



# Problem Statement

Last year, five randomly selected students took a math aptitude test before they began their statistics course. The Statistics Department has three questions.

- What linear regression equation best predicts statistics performance, based on math aptitude scores?
- If a student made an 80 on the aptitude test, what grade would we expect her to make in statistics?
- How well does the regression equation fit the data?

In the table below, the  $x_i$  column shows scores on the aptitude test. Similarly, the  $y_i$  column shows statistics grades. The last two rows show sums and mean scores that we will use to conduct the regression analysis.

	Student	$x_i$	$y_i$	$(x_i - \bar{x})$	$(y_i - \bar{y})$	$(x_i - \bar{x})^2$	$(y_i - \bar{y})^2$	$(x_i - \bar{x})(y_i - \bar{y})$
	1	95	85	17	8	289	64	136
	2	85	95	7	18	49	324	126
	3	80	70	2	-7	4	49	-14
	4	70	65	-8	-12	64	144	96
	5	60	70	-18	-7	324	49	126
<b>Sum</b>		390	385			730	630	470
<b>Mean</b>		78	77					

The regression equation is a linear equation of the form:  $\hat{y} = b_0 + b_1x$ . To conduct a regression analysis, we need to solve for  $b_0$  and  $b_1$ . Computations are shown below.

$b_1 = \Sigma [(x_i - \bar{x})(y_i - \bar{y})] / \Sigma [(x_i - \bar{x})^2]$ $b_1 = 470/730 = 0.644$	$b_0 = \bar{y} - b_1 * \bar{x}$ $b_0 = 77 - (0.644)(78) = 26.768$
--	---

Therefore, the regression equation is:  $\hat{y} = 26.768 + 0.644x$ .

Once you have the regression equation, using it is a snap. Choose a value for the independent variable ( $x$ ), perform the computation, and you have an estimated value ( $\hat{y}$ ) for the dependent variable.

In our example, the independent variable is the student's score on the aptitude test. The dependent variable is the student's statistics grade. If a student made an 80 on the aptitude test, the estimated statistics grade would be:

$$\hat{y} = 26.768 + 0.644x = 26.768 + 0.644 * 80 = 26.768 + 51.52 = 78.288$$

67

Warning: When you use a regression equation, do not use values for the independent variable that are outside the range of values used to create the equation. That is called extrapolation, and it can produce unreasonable estimates.

Whenever you use a regression equation, you should ask how well the equation fits the data. One way to assess fit is to check the coefficient of determination, which can be computed from the following formula.

$$R^2 = \{ ( 1 / N ) * \Sigma [ (x_i - \bar{x}) * (y_i - \bar{y}) ] / (\sigma_x * \sigma_y) \}^2$$

where N is the number of observations used to fit the model,  $\Sigma$  is the summation symbol,  $x_i$  is the x value for observation i,  $\bar{x}$  is the mean x value,  $y_i$  is the y value for observation i,  $\bar{y}$  is the mean y value,  $\sigma_x$  is the standard deviation of x, and  $\sigma_y$  is the standard deviation of y. Computations for the sample problem of this lesson are shown below.

$\sigma_x = \text{sqrt} [ \Sigma ( x_i - \bar{x} )^2 / N ]$ $\sigma_x = \text{sqrt}( 730/5 ) = \text{sqrt}(146) = 12.083$	$\sigma_y = \text{sqrt} [ \Sigma ( y_i - \bar{y} )^2 / N ]$ $\sigma_y = \text{sqrt}( 630/5 ) = \text{sqrt}(126) = 11.225$
$R^2 = \{ ( 1 / N ) * \Sigma [ (x_i - \bar{x}) * (y_i - \bar{y}) ] / ( \sigma_x * \sigma_y ) \}^2$ $R^2 = [ ( 1/5 ) * 470 / ( 12.083 * 11.225 ) ]^2 = ( 94 / 135.632 )^2 = ( 0.693 )^2 = 0.48$	

A coefficient of determination equal to 0.48 indicates that about 48% of the variation in statistics grades (the **dependent variable**) can be explained by the relationship to math aptitude scores (the **independent variable**). This would be considered a good fit to the data, in the sense that it would substantially improve an educator's ability to predict student performance in statistics class.

# Outliers and Influential Points

An outlier is an extreme observation that does not fit the general correlation or regression pattern. In the regression setting, outliers will be far away from the regression line in the y-direction. Since it is an unusual observation, the inclusion of an outlier may affect the slope and the y-intercept of the regression line.

When examining a scatterplot graph and calculating the regression equation, it is worth considering whether extreme observations should be included or not.

Let's use our example above to illustrate the effect of a single outlier. Say that we have a student who has a high GPA but who suffered from test anxiety the morning of the SAT verbal test and scored a 410. Using our original regression equation, we would expect the student to have a GPA of 2.2. But, in reality, the student has a GPA equal to 3.9. The inclusion of this value would change the slope of the regression equation from 0.0055 to 0.0032, which is quite a large difference.

There is no set rule when trying to decide whether or not to include an outlier in regression analysis. For univariate data, we can use the IQR rule to determine whether or not a point is an outlier. We should consider values that are 1.5 times the inter-quartile range below the first quartile or above the third quartile as outliers.

Extreme outliers are values that are 3.0 times the inter-quartile range below the first quartile or above the third quartile.

# Calculating Residuals and Understanding their Relation to the Regression Equation

Recall that the linear regression line is the line that best fits the given data. Ideally, we would like to minimize the distances of all data points to the regression line. These distances are called the error,  $e$ , and are also known as the residual values.

Student	SAT Score ( $X$ )	GPA ( $Y$ )	Predicted GPA ( $\hat{Y}$ )	Residual Value	Residual Value Squared
1	595	3.4	3.4	0	0
2	520	3.2	3.0	0.2	0.04
3	715	3.9	4.1	-0.2	0.04
4	405	2.3	2.3	0	0
5	680	3.9	3.9	0	0
6	490	2.5	2.8	-0.3	0.09
7	565	3.5	3.2	0.3	0.09
$\sum(y - \hat{y})^2$					0.26

A good line figure below that the residuals are the vertical distances between the observations and the predicted values on the regression line.



# Lab 4:

## Track and Field

## World Records



Every four years, track and field athletes take the world stage at the **Summer Olympics**. Some of the most exciting events during each Olympics are those in which athletes push the limits of their sport, breaking their own personal best records, national records, or even world records.

We have compiled the **world record times for track events** like the 100m dash and record distances for field events like the shotput into a single dataset.

This **dataset includes information on the person who broke the record, his/her nationality, where the record was broken, and the year it was broken**. Note that not all world records are broken during the Olympics, with many occurring in regional or national competitions

# PYTHON CODE

# Linear Regression

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
plt.scatter(x,y)

x = np.array(x).reshape((-1, 1))
y= np.array(y)

model.fit(x,y)

r_sq = model.score(x,y)
print("Variance for model",r_sq )
print('intercept model:', model.intercept_)
print('slope: model', model.coef_)
```

# MULTIPLE REGRESSION

- **Multiple linear regression** involves more than one predictor variable
- Training data is of the form  $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), \dots, (\mathbf{X}_{|D|}, y_{|D|})$
- where the  $\mathbf{X}_i$  are the  $n$ -dimensional training data with associated class labels,  $y_i$
- An example of a multiple linear regression model based on two predictor attributes:

$$y = w_0 + w_1x_1 + w_2x_2$$

# CPU Performance Data

	Cycle time (ns) MYCT	Main memory (KB)		Cache (KB) CACH	Channels		Performance PRP
		Min. MMIN	Max. MMAX		Min. CHMIN	Max. CHMAX	
1	125	256	6000	256	16	128	198
2	29	8000	32000	32	8	32	269
3	29	8000	32000	32	8	32	220
4	29	8000	32000	32	8	32	172
5	29	8000	16000	32	8	16	132
...							
207	125	2000	8000	0	2	14	52
208	480	512	8000	32	0	0	67
209	480	1000	4000	0	0	0	45

$$\text{PRP} = -55.9 + 0.0489 \text{ MYCT} + 0.0153 \text{ MMIN} + 0.0056 \text{ MMAX} \\ + 0.6410 \text{ CACH} - 0.2700 \text{ CHMIN} + 1.480 \text{ CHMAX}.$$

Various statistical measures exist for determining how well the proposed model can predict  $y$  (described later).

Obviously, the greater the number of predictor attributes is, the slower the performance is.

Before applying regression analysis, it is common to perform attribute subset selection to eliminate attributes that are unlikely to be good predictors for  $y$ .

In general, regression analysis is accurate for numeric prediction, except when the data contain outliers.

# Non linear regression

If data that does not show a linear dependence we can get a more accurate model using a **nonlinear regression** model

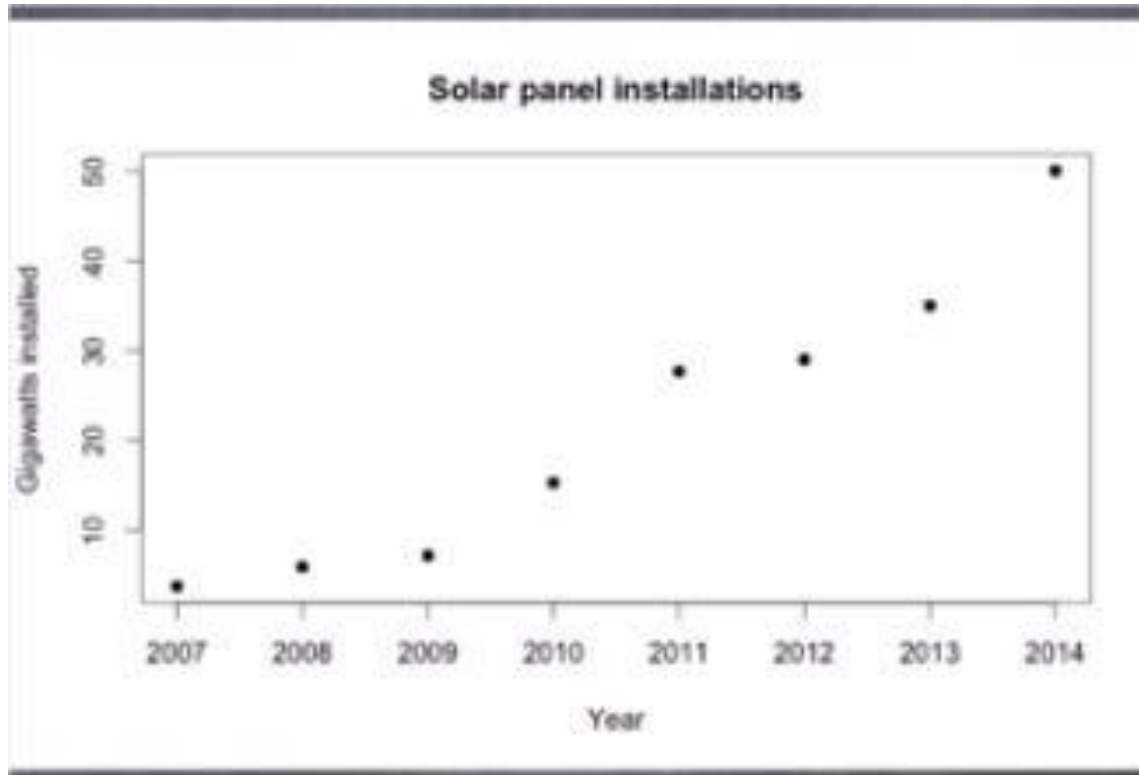
For example,

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

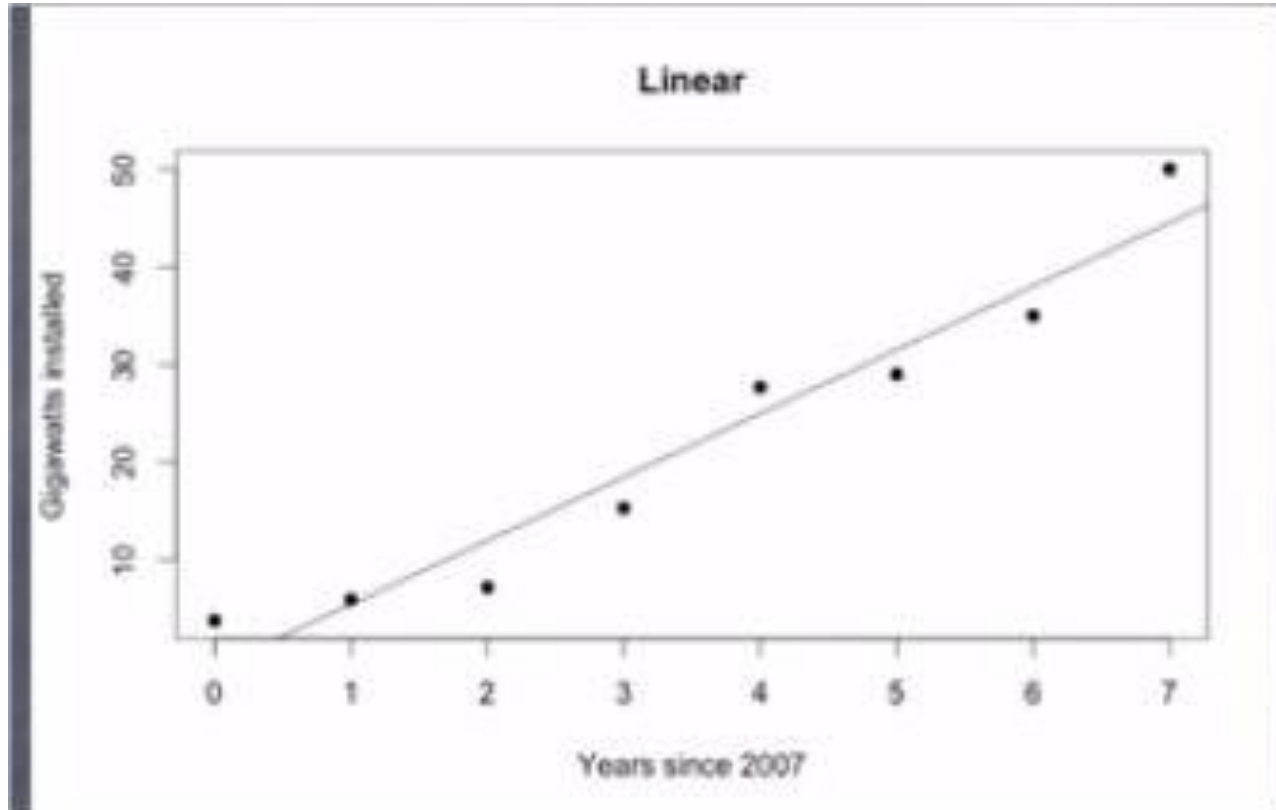


# EXPONENTIAL REGRESSION

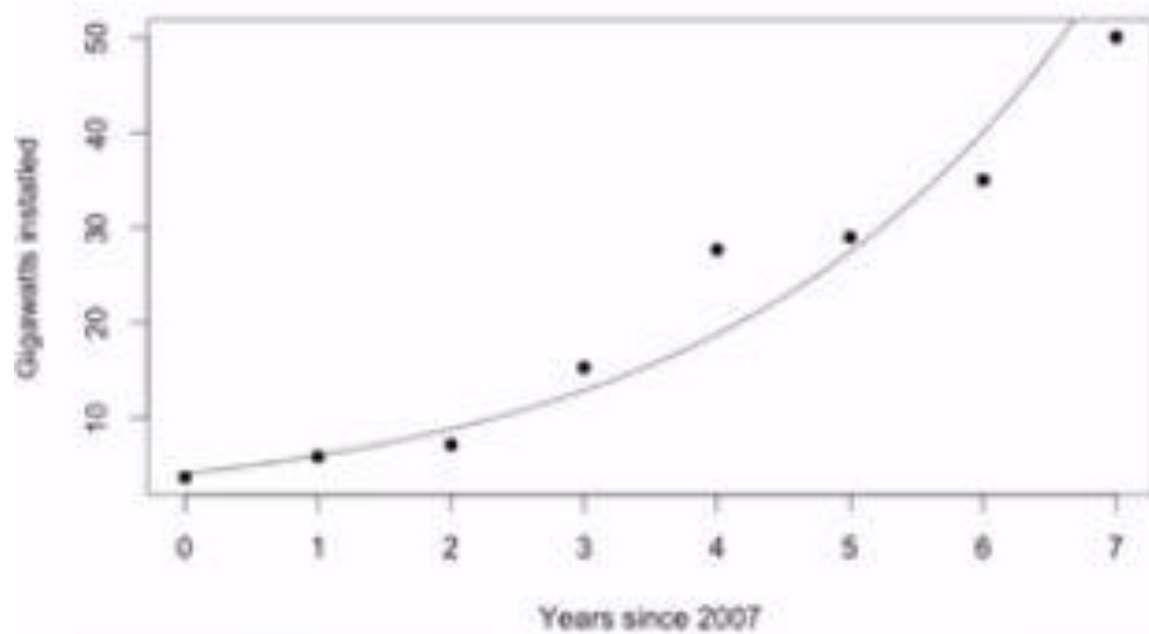
# No. of solar panel installations



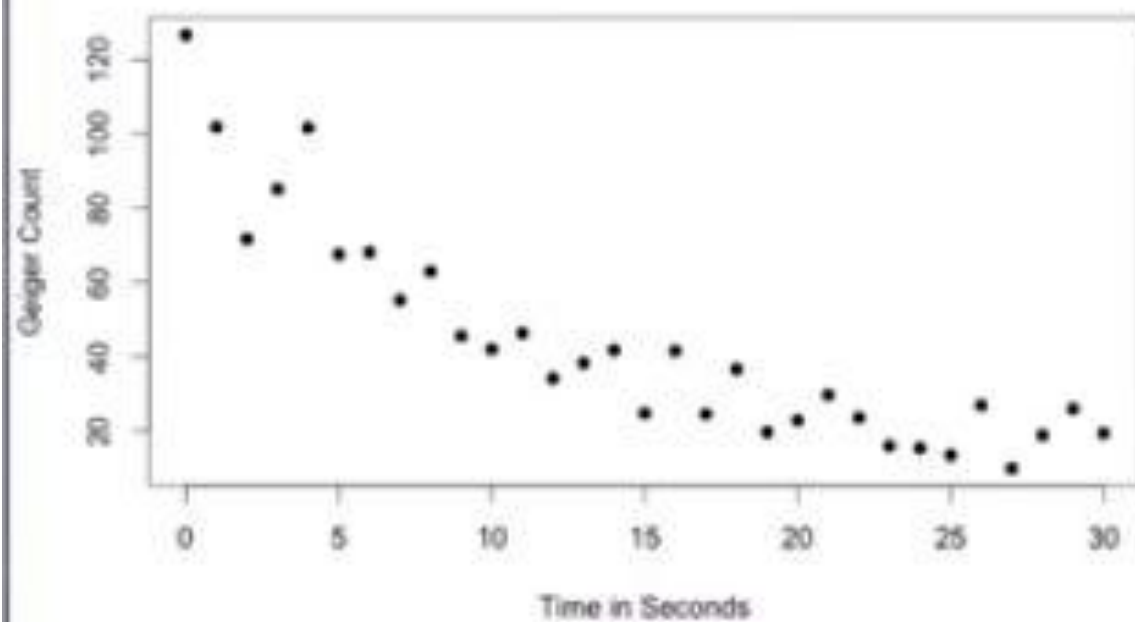
# No. of solar panel installations (Since 2007)



### Exponential



### Radioactivity



$$f(t) = a * b^t$$

Our "a" parameter is the value predicted for the outcome at year zero - much like our intercept to our linear model.

An exponential model has a constant PERCENT rate of change.

Let's get back to the interpretation of our "b" parameter.

The "b" parameter holds within it the percent rate of change. Formally, "b" is expressed as  $1+r$ , with  $r$  being the proportional change of the outcome variable for every whole unit increase on the input variable.

An "a" parameter instead of an intercept, but the interpretation is pretty much the same.

And a "b" parameter of a growth factor which has the growth rate or "r" as part of its make up - and we can sort of think of this as a "slope" to the model.

When "b" is above 1, we have growth, and when "b" is less than 1, we have decay.

And the biggest, clearest, difference between exponential and linear models:

a constant rate of change for the linear, **but a constant "percent" rate of change for the exponential model.**

## EXAMPLE I

You have a cylinder that is filled with water to a height of 50 centimeters. The cylinder has a hole at the bottom which is covered with a stopper. The stopper is released at time  $t = 0$  seconds and allowed to empty. The following data shows the height of the water in the cylinder at different times.

Time(sec)	0	2	4	6	8	10	12	14	16	18	20	22	24
Height(cm)	50	42.5	35.7	29.5	23.8	18.8	14.3	10.5	7.2	4.6	2.5	1.1	0.2

- Find the height (in centimeters) of water in the cylinder as a function of time in seconds.
- Find the height of the water when  $t = 5$  seconds.
- Find the height of the water when  $t = 13$  seconds.



Begin by graphing to get a visual image of what the relationship looks like. Let's begin by defining the variables.

Define  $x$  = the time in seconds

$y$  = height of the water in centimeters

Notice that most of the points seem to fit on a straight line when the water level is high. Assume that a

function relating the height of the water to the time is

*Now Solve for the Equation of the Line:*  
linear.

We will find the equation of the line from two data points from the time/height data.

We have selected  $(0, 50)$  and  $(4, 35.7)$ , as these two points fall very close to the line.

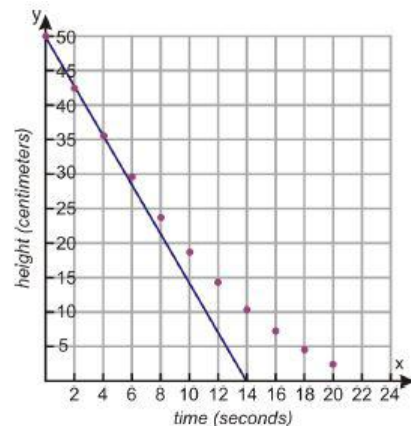
Using the formula  $y = mx + b$ , we begin by solving for the slope ( $m$ ) and then for the  $y$ -intercept ( $b$ ).

$$m = \frac{-14.3}{4} = -3.58$$

We obtain the function:  $y = -3.58x + 50$

$$35.7 = -3.58(4) + b$$

$$b = 50$$



The height of the water when  $t = 5$  seconds is

$$y = -3.58(5) + 50 = 32.1 \text{ cm}$$

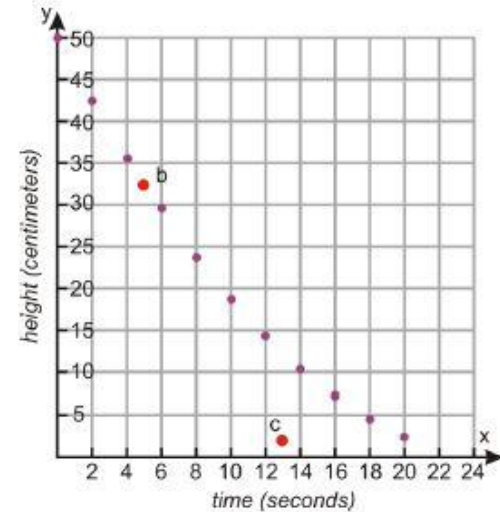
The height of the water when  $t = 13$  seconds is

$$y = -3.58(13) + 50 = 3.46 \text{ cm}$$

### Check

To check the validity of the solutions, let's plot the answers to on the scatter plot. We see that the answer to b) is close to the rest of the data, but the answer to c) does not seem to follow the linear trend.

We can conclude that when the water level is high, the relationship between the height of the water and the time is a linear function. When the water level is low, we must change our assumption. There must be a non-linear relationship between the height and the time.



### *Solve with new assumptions*

If you are familiar with quadratic equations, you might recognize the form here and use a graphing calculator or other tool to help you find the equation of the function.

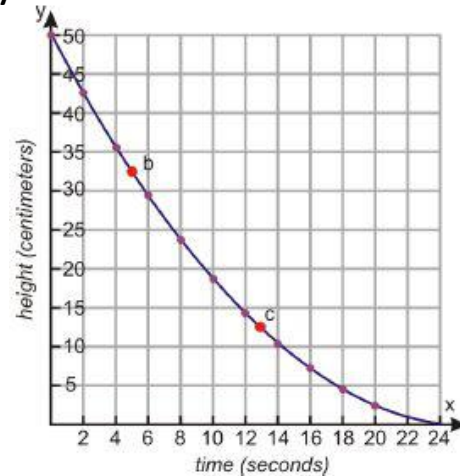
a. We obtain the function  $y = 0.075x^2 - 3.87x + 50$

b. The height of the water when  $t = 5$  seconds is  $y = .075(5)^2 - 3.87(5) + 50 = 32.53$  centimeters

c. The height of the water when  $t = 13$  seconds is  $y = :075(13)^2 - 3.87(13) + 50 = 12.37$  centimeters

### *Check*

To check the validity of the solutions lets plot the answers to b) and c) on the scatterplot. We see that the answer to both b) and c) are close to the rest of the data. We conclude that a quadratic function represents the situation more accurately than a linear function. However, for high water levels the linear function is an equally good representation.

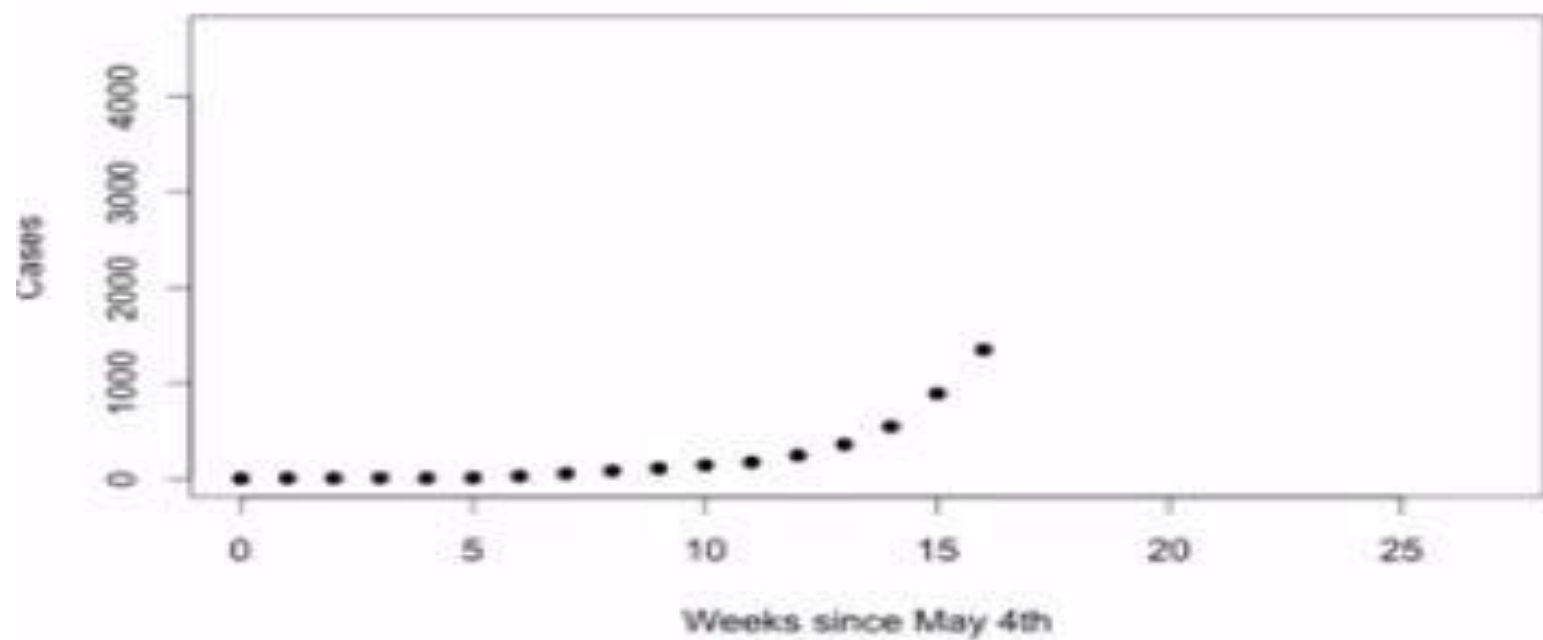


92

# LOGISTIC REGRESSION

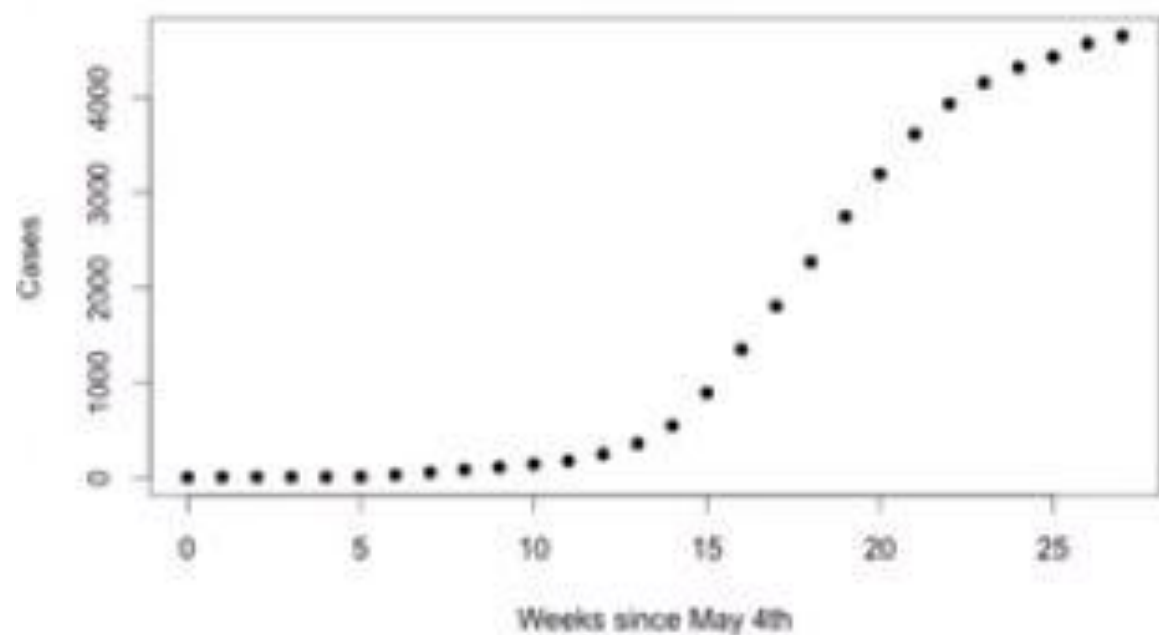
---

**Cumulative ebola cases  
Liberia**



---

Cummulative ebola cases  
Liberia



## What is it?

We don't have resources to continue with exponential growth and have an upper limit

That's where the logistic growth function comes in.

Instead of seeing this - an exponential model of zombie cell growth - we're more likely to see something like this - logistic growth.

$$f(t) = \frac{C}{1 + ab^{-t}}$$

C = upper limit

Now to get to that limit, the curve of our model needs to "turn over" - and we call the point at which it turns over the **"inflection point"** - defined as the Carrying Capacity divided by 2. ( $C/2$ )

Now, we can use our log rules to find out where on our input variable that inflection point happens. Or we can simply use the formula of  $\log(a) / \log(b)$ .

The **"a" parameter** is a **"helper."** It's used to help define where the  $f(t)$  is when our input variable is at zero.

Solving for an input value of zero, we see that the logistic function reduces to  $C / (1+a)$ .

The **"b" parameter** is the same. It looks like "b" from exponential, but it's not. In the case of **logistic growth**, **b must be greater than 1.**



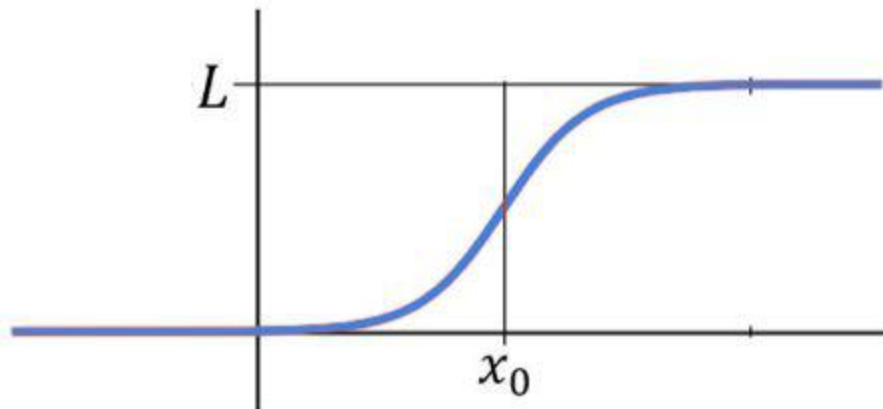
# Logistic Function

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}}$$

$x_0$  = x value of midpoint

$L$  = maximum value

$k$  = growth rate



## Example 2

A scientist counts 2,000 fish in a lake. The fish population increases at a rate of 1.5 fish per generation but the lake has space and food for only 2,000,000 fish. The table gives the number of fish (in thousands) in each generation.

Generation	0	4	8	12	16	20	24	28
Number (thousands)	2	15	75	343	1139	1864	1990	1999

- Find the number of fish as a function of generation.
- Find the number of fish in generation 10.
- Find the number of fish in generation 25.

## ***Solution***

We will define the variables and graph the relationship again.

Define  $x$  = the generation number and  $y$  = the number of fish in the lake.

We know that a population can increase exponentially. So, maybe we assume that we can use an exponential function to describe the relationship between the generation number and the number of fish.

Solve

- Since the population increases at a rate of 1.5 per generation, assume the function  $y = 2(1.5)^x$
- The number of fish in generation 10 is:  $y = 2(1.5)^{10} = 115$  thousand fish
- The number of fish in generation 25 is:  $y = 2(1.5)^{25} = 50502$  thousand fish

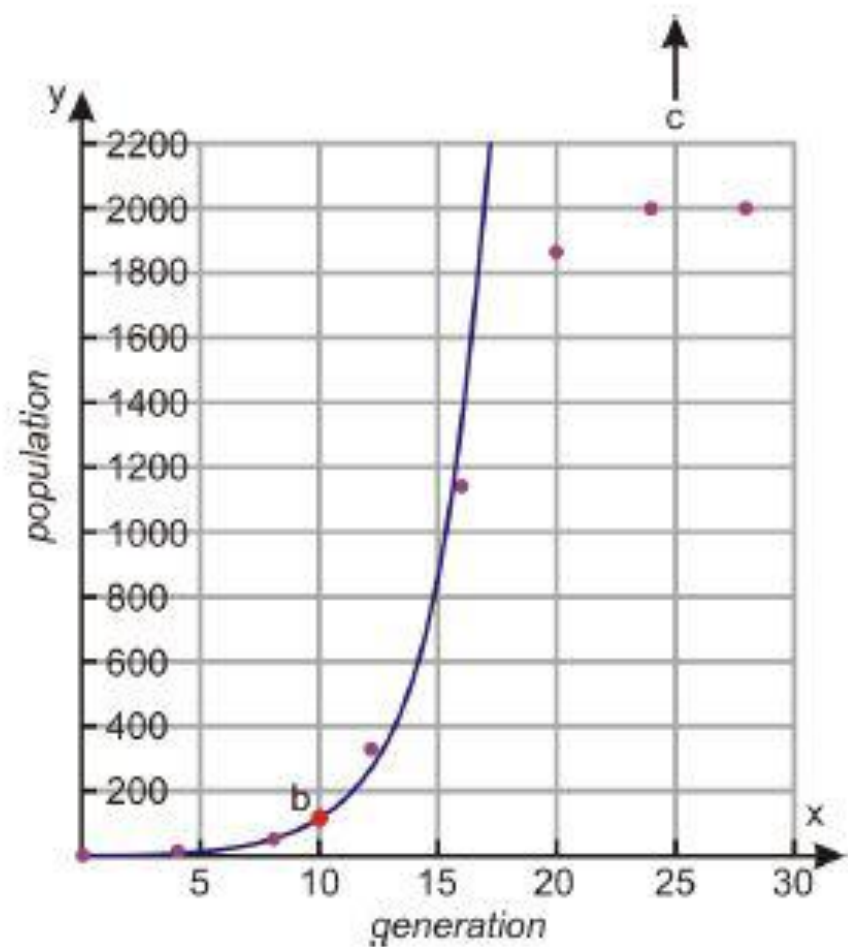
## Check

To check the validity of the solutions, let's plot the answers to b) and c) on the scatter plot (see the red dots).

We see that the answer to b) fits the data well but the answer to c) does not seem to follow the trend very closely.

The result is not even on our graph!

When the population of fish is high, the fish compete for space and resources so they do



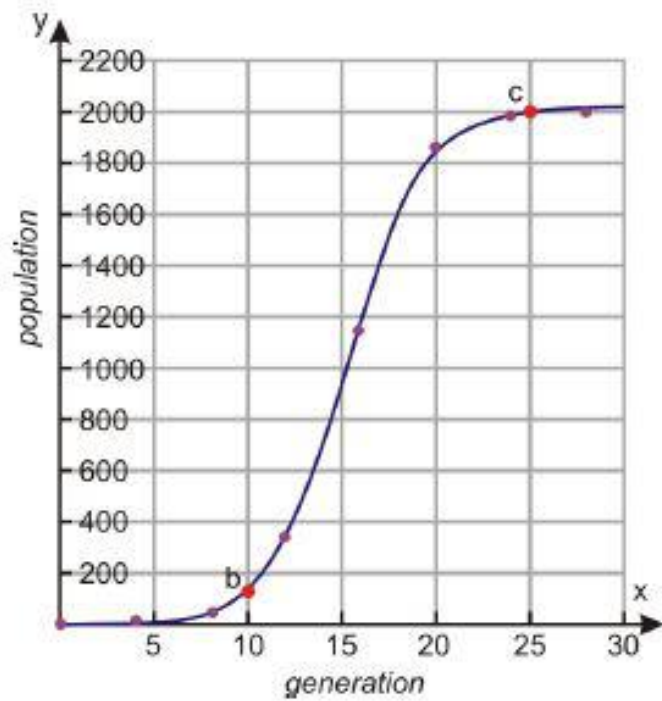
# Logistic Approach

*Solve with new assumptions:* When we try different models, we find that a logistic function fits the data the best.

a. We obtain the function  $y = \frac{2023.6}{1+1706.3(2.71)^{-.484x}}$

b. The number of fish in generation 10 is  $y = \frac{2023.6}{1+1706.3(2.71)^{-.484(10)}} = 139.6$  thousand fish

c. The number of fish in generation 25 is  $y = \frac{2023.6}{1+1706.3(2.71)^{-.484(25)}} = 2005$  thousand fish



We conclude that a logistic function represents the situation more accurately than an exponential function. However, for small populations the exponential function is an equally good representation, and it is much easier to

# Linear v/s Logistic Regression

---

- Consider the following example
- If X contains the area in square feet of houses, and Y contains the corresponding sale price of those houses
  - **Linear regression** - to predict selling price as a function of house size. The output will be the price value
  - **Logistic regression** - to predict, whether a house would sell for more than \$200K, The possible outputs are either Yes, the house will sell for more than \$200K, or No, the house will not

Linear Regression	Logistic Regression
In linear regression, the outcome (dependent variable) is continuous. It can have any one of an infinite number of possible values	In logistic regression, the outcome (dependent variable) has only a limited number of possible values. Mostly two



## Logistic Regression

---

- Logistic Regression produces results in a binary format
- It is used for predicting the outcome of a categorical dependent variable based on one or more features
- It is most widely used when the dependent variable is binary i.e., the number of available categories is two
- Some of the usual outputs of logistic regression are,
  - Yes and No
  - True and False
  - High and Low
  - Pass and Fail

## Use Case – Hands On

- A car company has released a new suv in the market, using the data they have about the earlier sales of their suv's they want to predict what type of people will be interested in buying this
- The dataset contains

User ID	Gender	Age	EstimatedSalary	Purchased
15624510	Male	19	19000	0
15810944	Male	35	20000	0
15668575	Female	26	43000	0
15603246	Female	27	57000	0
15804002	Male	19	76000	0
15728773	Male	27	58000	0
15598044	Female	27	84000	0
15694829	Female	32	150000	1
15600575	Male	25	33000	0
15727311	Female	35	65000	0

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
dataset2 = pd.read_csv("cars")
```

```
X = dataset2.iloc[:, [2,3]].values
```

```
Y = dataset2.iloc[:,4].values
```

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,  
test_size=0.25, random_state=0)
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.fit_transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
```

```
classifier = LogisticRegression(random_state=0)
```

```
classifier.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, l1_ratio=None, max_iter=100,  
                    multi_class='auto', n_jobs=None, penalty='l2',  
                    random_state=0, solver='lbfgs', tol=0.0001, verbose=0,  
                    warm_start=False)
```

```
Y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(Y_test, Y_pred)
```

```
print(cm)
```

```
[[63  5]  
 [ 8 24]]
```

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

**LET'S SEE IF GENDER PLAYS A  
ROLE**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
dataset2 = pd.read_csv("cars")
```

```
X =dataset2.iloc[:, [1,2,3]].values
```

```
Y = dataset2.iloc[:,4].values
```

```
Xgender = X[:,0]
```

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

```
Xgender= LabelEncoder().fit_transform(Xgender)
```

```
onen = OneHotEncoder(categories = 'auto')
```



```
onen = OneHotEncoder(categories = 'auto')
```

```
xg = pd.DataFrame(Xgender)
```

xg

	0
0	1
1	1
2	0
3	0
4	1

```
onen = OneHotEncoder(categories = 'auto')
```

```
a= onen.fit_transform(xg).toarray()
```

```
df = pd.DataFrame(data=a, columns=["Male", "Female"])
```

```
df
```

	Male	Female
<b>0</b>	0.0	1.0
<b>1</b>	0.0	1.0
<b>2</b>	1.0	0.0
<b>3</b>	1.0	0.0

```
In [27]: X
```

```
Out[27]: array([[ 'Male', 19, 19000],  
               [ 'Male', 35, 20000],  
               [ 'Female', 26, 43000],  
               ...,  
               [ 'Female', 50, 20000],  
               [ 'Male', 36, 33000],  
               [ 'Female', 49, 36000]], dtype=object)
```

```
In [28]: X = X[:, [1,2]]
```

```
In [30]: X = pd.DataFrame(X)
```

```
In [31]: X
```

```
Out[31]:
```

**0**

**1**

---

400 rows × 2 columns

```
XF = pd.concat([X, df], axis=1)
```

XF

	0	1	Male	Female
0	19	19000	0.0	1.0
1	35	20000	0.0	1.0
2	26	43000	1.0	0.0
3	27	57000	1.0	0.0
4	19	76000	0.0	1.0

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(XF,Y, test_size  
=0.25, random_state =0)
```

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.fit_transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
```

```
classifier = LogisticRegression(random_state=0)
```

```
classifier.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, l1_ratio=None, max_iter=100,  
                    multi_class='auto', n_jobs=None, penalty='l2',  
                    random_state=0, solver='lbfgs', tol=0.0001, verbose=0)
```

```
Y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(Y_test, Y_pred)
```

```
print(cm)
```

```
[[64  4]  
 [ 5 27]]
```

# Model Performance



Model performance parameters are metrics used to evaluate the performance of a machine learning model.

These metrics help to determine the accuracy and reliability of a model, and provide a way to compare different models and choose the best one for a given task.

We can use a confusion matrix to evaluate our model.  
For example, imagine testing for disease.

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

Example: Test for presence of disease  
NO = negative test = False = 0  
YES = positive test = True = 1

# Confusion Matrix

n=165		Predicted: NO	Predicted: YES	
Actual: NO		TN = 50	FP = 10	60
Actual: YES		FN = 5	TP = 100	105
		55	110	

## Basic Terminology:

- True Positives (TP)
- True Negatives (TN)
- False Positives (FP)
- False Negatives (FN)

n=165		Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60	
Actual: YES	FN = 5	TP = 100	105	
	55	110		

Accuracy:

- Overall, how often is it **correct**?
- $(TP + TN) / \text{total} = 150/165 = 0.91$

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Misclassification Rate  
(Error Rate):

- Overall, how often is it **wrong**?
- $(FP + FN) / \text{total} = 15/165 = 0.09$

## Actual Values

Positive (1)    Negative (0)

Predicted Values

Positive (1)

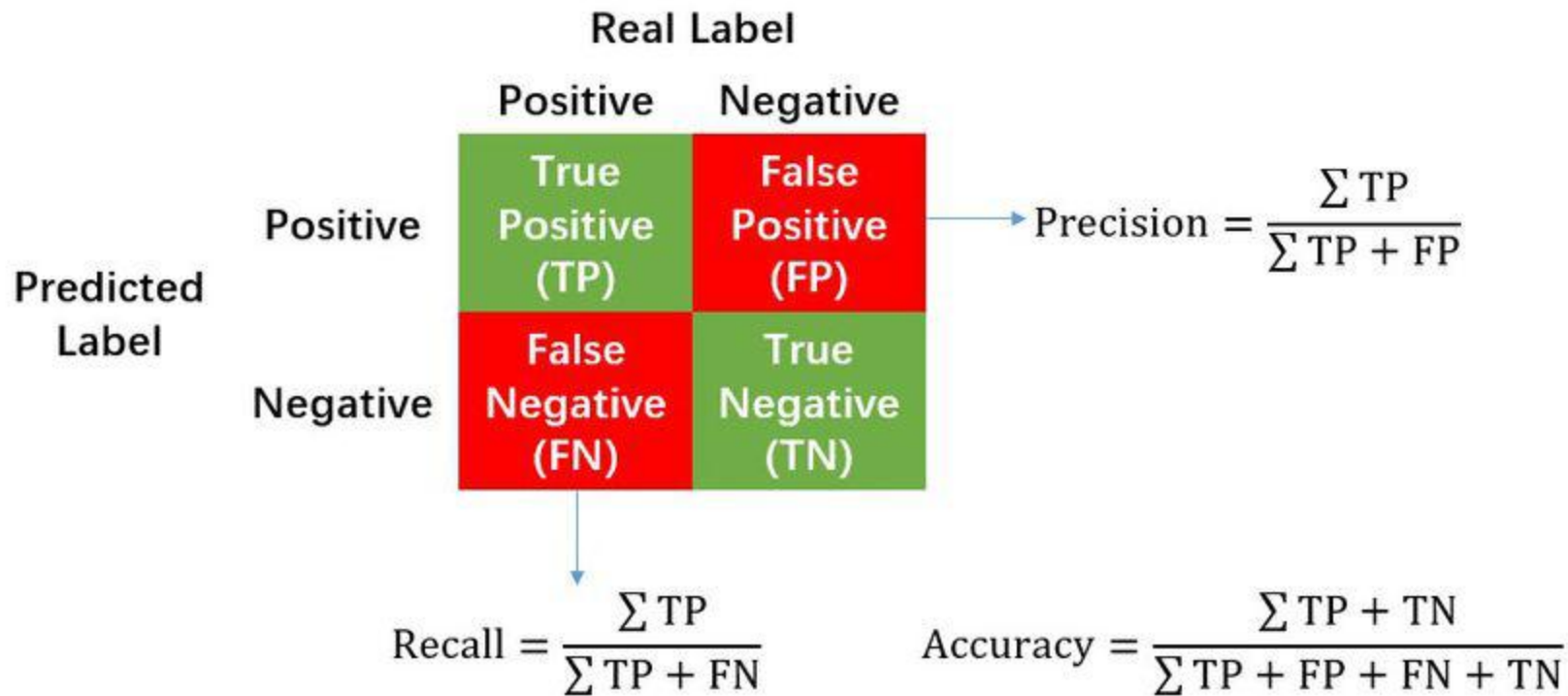
TP

FP

Negative (0)

FN

TN



# Metrics

- **Accuracy:** Accuracy is the fraction of correctly classified samples out of the total number of samples. It is a commonly used metric for classification problems and is expressed as a percentage.
- **Precision:** Precision is the fraction of true positive predictions out of all positive predictions. It measures how many of the positive predictions made by the model are actually true.
- **Recall (Sensitivity):** Recall is the fraction of true positive predictions out of all actual positive cases. It measures how well the model is able to detect positive cases.
- **F1 Score:** The F1 score is the harmonic mean of precision and recall. It provides a single metric that balances precision and recall and is useful when the positive class is rare.



# Metrics (contd.)

- **Confusion Matrix:** A confusion matrix is a table that summarizes the performance of a binary classifier. It shows the number of true positive, false positive, false negative, and true negative predictions made by the model.
- **ROC Curve:** The ROC (receiver operating characteristic) curve is a plot of the true positive rate versus the false positive rate for different thresholds. It provides a visual representation of the trade-off between precision and recall and is useful for comparing different classifiers.
- **AUC (Area under the ROC Curve):** The AUC is the area under the ROC curve and provides a single scalar metric that summarizes the performance of a classifier. A higher AUC indicates a better-performing classifier.
- **Mean Absolute Error (MAE):** Mean absolute error is the average absolute difference between the predicted and actual values. It is commonly used for regression problems.
- **Mean Squared Error (MSE):** Mean squared error is the average squared difference between the predicted and actual values. It is a commonly used metric for regression problems, but can be sensitive to outliers.

**LET'S SEE IF GENDER PLAYS A  
ROLE**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
dataset2 = pd.read_csv("cars")
```

```
X =dataset2.iloc[:, [1,2,3]].values
```

```
Y = dataset2.iloc[:,4].values
```

```
Xgender = X[:,0]
```

```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder
```

```
Xgender= LabelEncoder().fit_transform(Xgender)
```

```
onen = OneHotEncoder(categories = 'auto')
```

```
onen = OneHotEncoder(categories = 'auto')
```

```
xg = pd.DataFrame(Xgender)
```

xg

	0
0	1
1	1
2	0
3	0
4	1

```
onen = OneHotEncoder(categories = 'auto')
```

```
a= onen.fit_transform(xg).toarray()
```

```
df = pd.DataFrame(data=a, columns=["Male", "Female"])
```

```
df
```

	Male	Female
<b>0</b>	0.0	1.0
<b>1</b>	0.0	1.0
<b>2</b>	1.0	0.0
<b>3</b>	1.0	0.0

```
In [27]: X
```

```
Out[27]: array([[ 'Male', 19, 19000],  
                [ 'Male', 35, 20000],  
                [ 'Female', 26, 43000],  
                ...,  
                [ 'Female', 50, 20000],  
                [ 'Male', 36, 33000],  
                [ 'Female', 49, 36000]], dtype=object)
```

```
In [28]: X = X[:, [1,2]]
```

```
In [30]: X = pd.DataFrame(X)
```

```
In [31]: X
```

```
Out[31]:
```

**0**

**1**

---

400 rows × 2 columns

```
XF = pd.concat([X, df], axis=1)
```

XF

	0	1	Male	Female
0	19	19000	0.0	1.0
1	35	20000	0.0	1.0
2	26	43000	1.0	0.0
3	27	57000	1.0	0.0
4	19	76000	0.0	1.0

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, Y_train, Y_test = train_test_split(XF,Y, test_size  
=0.25, random_state =0)
```



```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

```
X_train = sc.fit_transform(X_train)
```

```
X_test = sc.fit_transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
```

```
classifier = LogisticRegression(random_state=0)
```

```
classifier.fit(X_train, Y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,  
                    intercept_scaling=1, l1_ratio=None, max_iter=100,  
                    multi_class='auto', n_jobs=None, penalty='l2',  
                    random_state=0, solver='lbfgs', tol=0.0001, verbose=0)
```

```
Y_pred = classifier.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
```

```
cm = confusion_matrix(Y_test, Y_pred)
```

```
print(cm)
```

```
[[64  4]  
 [ 5 27]]
```

# Overfitting

Overfitting and underfitting are two common problems in machine learning that occur when a model is either too complex or too simple for the data it is trained on.

Overfitting occurs when a model is too complex and has learned the noise in the data instead of the underlying relationship between the inputs and outputs. As a result, the model performs well on the training data but poorly on new, unseen data. Overfitting can be thought of as a model that has memorized the training data, but has not learned to generalize to new data.

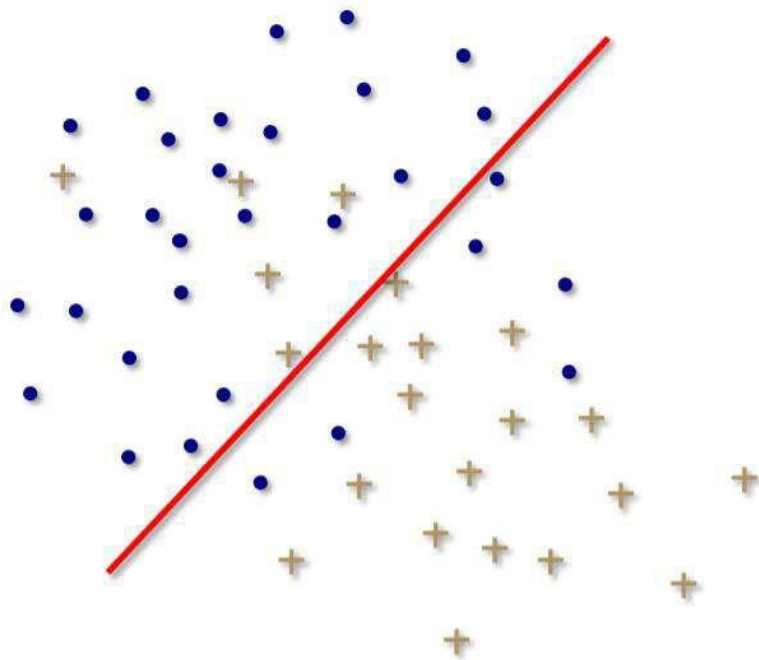
# Underfitting

Underfitting occurs when a model is too simple and cannot capture the underlying relationship between the inputs and outputs. As a result, the model performs poorly on both the training data and new, unseen data. Underfitting can be thought of as a model that has not learned the pattern in the data.

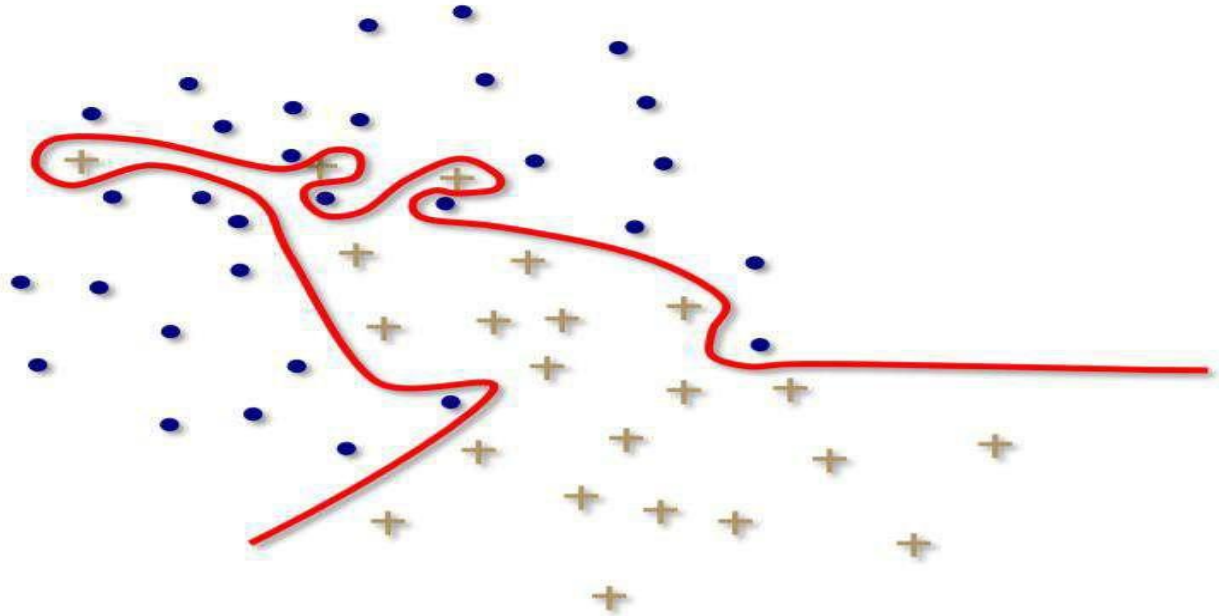
In both cases, it is important to find the right balance between model complexity and simplicity. A model that is too complex will overfit the data, while a model that is too simple will underfit the data. The goal is to find a model that is complex enough to capture the underlying relationship in the data, but not so complex that it memorizes the noise in the data.

# Overfitting

---



# Overfitting

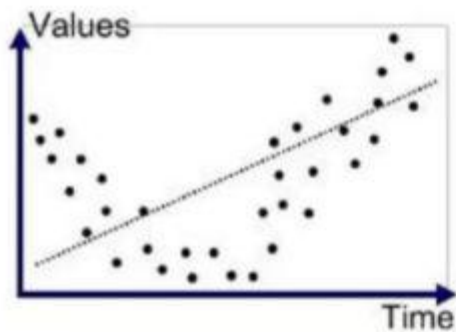


# Model (Function) Fitting

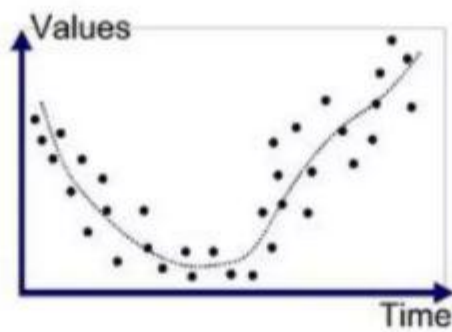
- How well a model performs on training/evaluation datasets will define its characteristics

	Underfit	Overfit	Good Fit
Training Dataset	Poor	Very Good	Good
Evaluation Dataset	Very Poor	Poor	Good

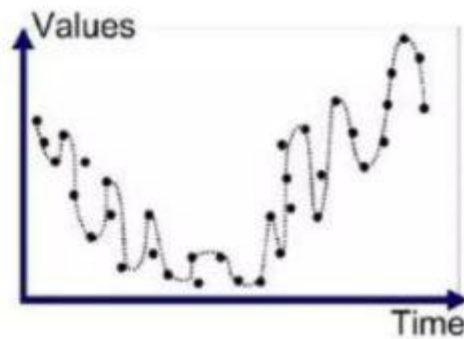
# Model Fitting – Visualization



Underfitted



Good Fit/Robust



Overfitted

Variations of model fitting [1]



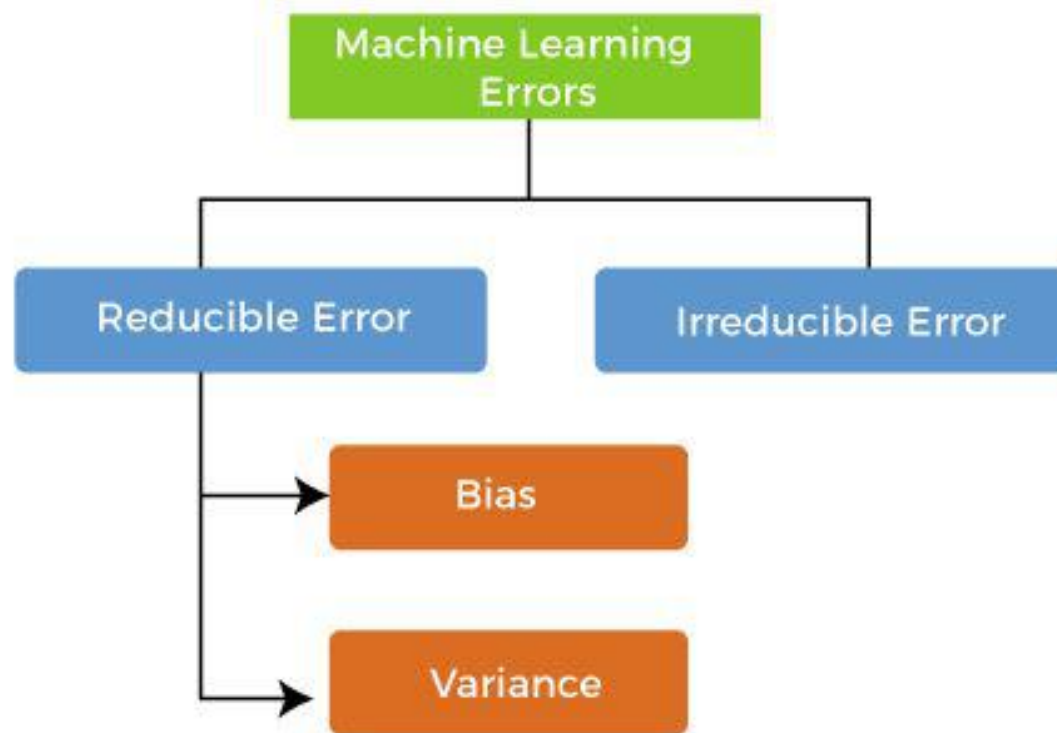
# Bias Variance

- **Bias**

- Represents the extent to which average prediction over all data sets differs from the desired regression function

- **Variance**

- Represent the extent to which the model is sensitive to the particular choice of data set



# Bias and Variance

In machine learning, variance and bias are two important concepts that describe the errors that can occur in a model.

**Bias** refers to the **difference between the predicted values of a model and the true values**. A model with high bias tends to consistently predict the same value, regardless of the input. This can happen when the model is too simple and doesn't have enough capacity to capture the underlying relationships in the data. High bias can lead to underfitting, where the model is not flexible enough to fit the training data well.

**Variance**, on the other hand, refers to the **variability of a model's predictions for a given input**. A model with high variance is sensitive to small fluctuations in the training data and will produce different predictions for similar inputs. This can lead to overfitting, where the model becomes too complex and fits the training data too well, but performs poorly on unseen data.

# Example

Suppose you are building a model to predict the height of a person based on their weight.

A model with **high bias** might make a very simple assumption, such as: "**every person is exactly 5 feet tall.**" This model would have a **low variance**, because it would give the same prediction for any input. However, it would also have high bias, because it would **consistently overestimate or underestimate the true height** of a person.

On the other hand, a model with **high variance** might make a **very complex assumption**, such as: "**the height of a person is equal to their weight multiplied by a random value.**" This model would have a high variance, because the randomness in the model would lead to **different predictions for similar inputs**. However, it would have low bias, because it would fit the training data very well, but would likely perform poorly on unseen data.

A good model should strive to find the **right balance between bias and variance**, so that it generalizes well to unseen data, while still fitting the training data accurately.

# How to reduce Bias?

There are several ways to reduce bias in machine learning models:

1. **Increasing model complexity:** By adding more parameters, hidden layers, or increasing the capacity of the model, you can reduce the bias. This can be done by using more complex algorithms like deep neural networks.
2. **Adding more features:** Adding relevant features to the model can help capture more of the underlying relationship between inputs and outputs, which can reduce bias.
3. **Increasing the size of the training data:** With more data, a model can learn the true relationship between inputs and outputs, which can reduce bias.
4. **Regularization:** Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function. This term discourages the model from assigning too much importance to any one feature or from making the model too complex. Some common forms of regularization are L1 regularization, L2 regularization, and dropout.
5. **Cross-validation:** Cross-validation is a technique used to evaluate a model's performance by splitting the data into training and validation sets. By evaluating the model on the validation set, you can get a better idea of how well the model will perform on unseen data, which can help reduce overfitting and bias.

# How to reduce Variance?

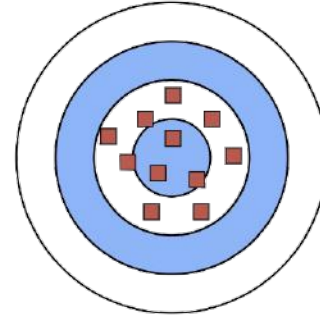
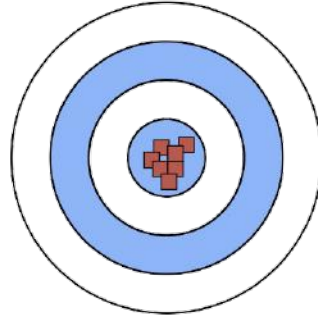
There are several ways to reduce variance in machine learning models:

1. **Simplifying the model:** By reducing the number of parameters, hidden layers, or reducing the capacity of the model, you can reduce the variance. This can be done by using simpler algorithms like linear regression or decision trees.
2. **Feature selection:** By removing irrelevant or redundant features from the model, you can reduce the variance caused by noise in the data.
3. **Ensemble methods:** Ensemble methods are techniques that combine the predictions of multiple models to produce a more robust prediction. This can reduce variance by combining the strengths of multiple models and reducing the impact of any one model's weaknesses. Examples of ensemble methods include random forests, gradient boosting, and bagging.
4. **Early stopping:** Early stopping is a technique used to prevent overfitting by stopping the training process when the model's performance on a validation set starts to degrade. By stopping the training early, you can prevent the model from learning the noise in the data, which can reduce variance.
5. **Regularization:** Regularization is a technique used to prevent overfitting by adding a penalty term to the loss function. This term discourages the model from assigning too much importance to any one feature or from making the model too complex. Some common forms of regularization are L1 regularization, L2 regularization, and dropout.

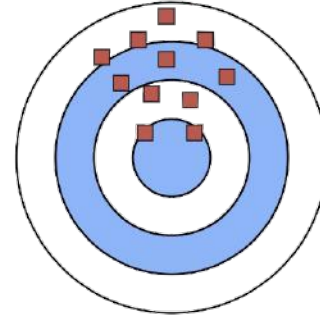
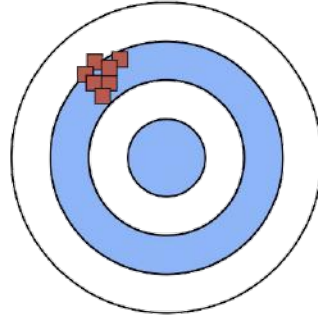
**Low Variance**  
(Precise)

**High Variance**  
(Not Precise)

**Low Bias**  
(Accurate)



**High Bias**  
(Not Accurate)

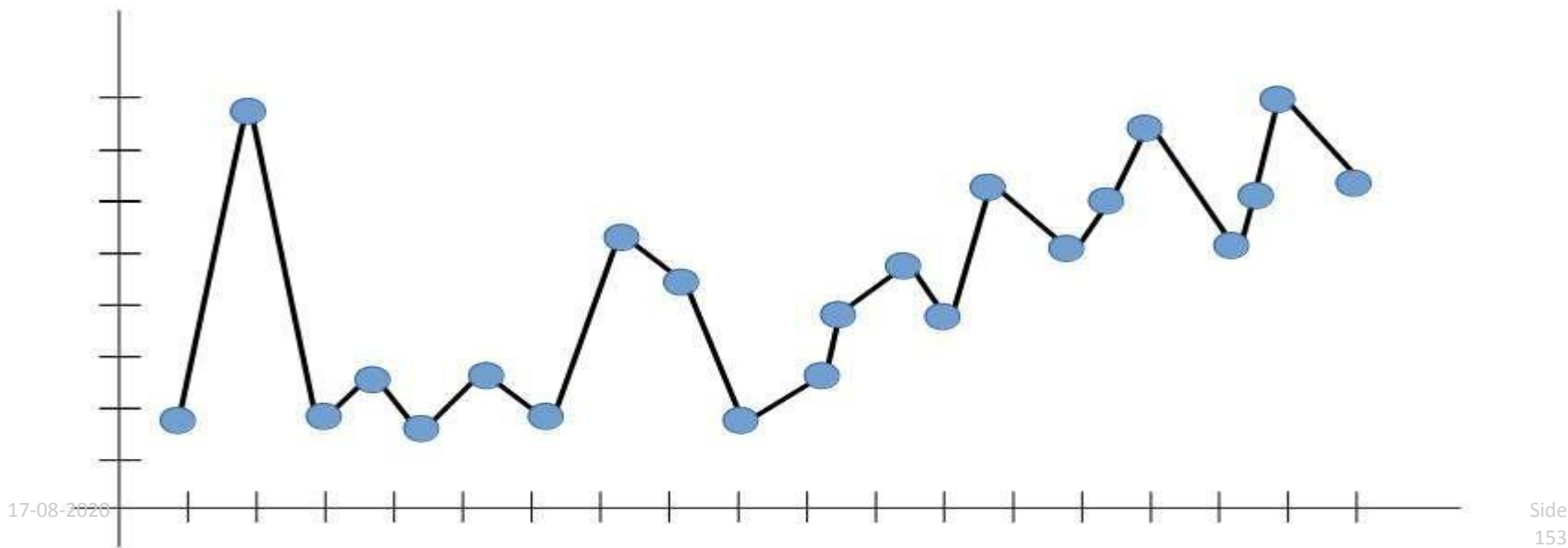


# Generalization and Over Fitting

- Generalization is the model's ability to give sensible outputs to sets of input that it has never seen before
- Arguably, Machine Learning models have one sole purpose; to generalize well
- A model that generalizes well is a model that is neither underfit nor overfit
- When we run our training algorithm on the data set, we allow the overall cost (i.e. distance from each point to the line) to become smaller with more iterations. Leaving this training algorithm run for long leads to minimal overall cost. However, this means that the line will be fit into all the points (including noise), catching secondary patterns that may not be needed for the generalizability of the model
- If we leave the learning algorithm running for long, it could end up fitting the line in the following manner:



# Generalization and Over Fitting



- This looks good, right? Yes, but is it reliable? Well, not really

# Generalization and Over Fitting

- In the figure above, the algorithm captured all trends — but not the dominant one. If we want to test the model on inputs that are beyond the line limits we have (i.e. generalize), what would that line look like? There is really no way to tell. Therefore, the outputs aren't reliable
- If the model does not capture the dominant trend that we can all see (positively increasing, in our case), it can't predict a likely output for an input that it has never seen before — defying the purpose of Machine Learning to begin with

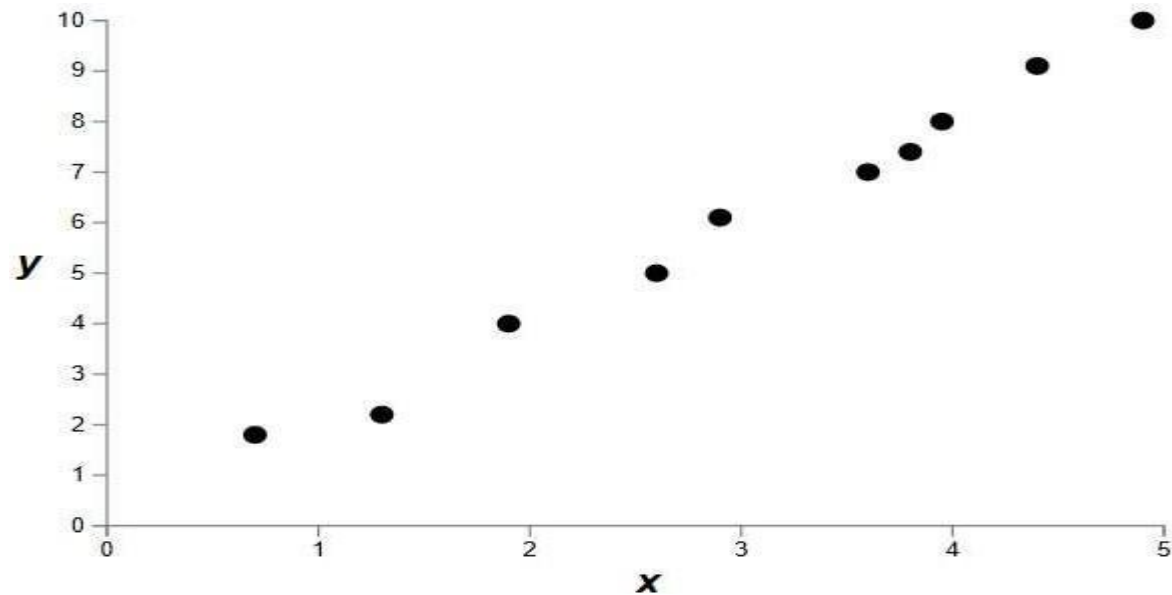
# Regularization

- Too many parameters = overfitting
  - Not enough parameters = underfitting
  - More data = less chance to overfit
  - How do we know what is required?
-

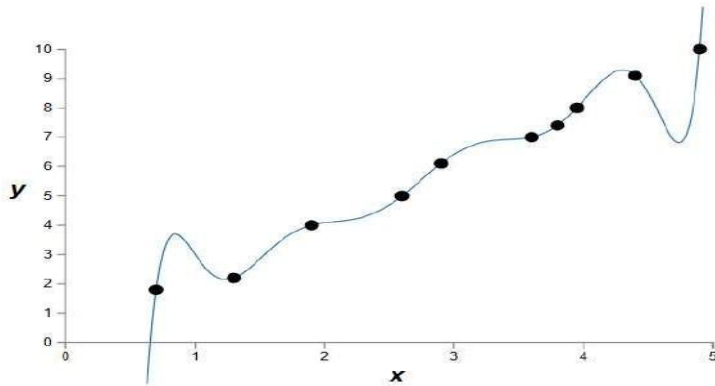
# Regularization

- Attempt to guide solution to not overfit
  - But still give freedom with many parameters
-

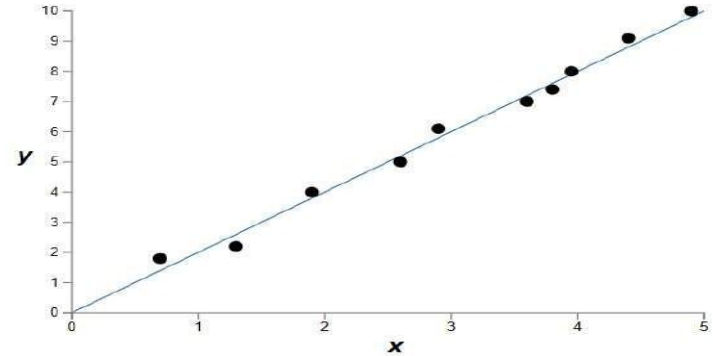
# Data fitting problem



# Which is better a priori?



9<sup>th</sup> order polynomial



1<sup>st</sup> order polynomial

# Regularization

- Attempt to guide solution to not overfit
- But still give freedom with many parameters
- Idea:  
Penalize the use of parameters to prefer small weights.
- Idea: add a cost to having high weights
- $\lambda$  = regularization parameter

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

# Forms of Regularization

- Adding more data is a kind of regularization
  - Pooling is a kind of regularization
  - Data augmentation is a kind of regularization
-



# Regularization

L1, L2, and L3 regularization are different types of regularization that are used to prevent overfitting in machine learning models.

**L1 Regularization:** Also known as **Lasso regularization**, L1 regularization adds a **penalty term to the loss function that is proportional to the absolute value of the coefficients**. This leads to sparse solutions, where many of the coefficients are exactly zero. This can be **useful for feature selection**, as it effectively removes the least important features from the model.

# L2 Regularization

Also known as **Ridge regularization**, L2 regularization **adds a penalty term to the loss function that is proportional to the square of the coefficients**. This leads to **small coefficients, which can prevent overfitting**. The effect of L2 regularization is to shrink the coefficients towards zero, but not to zero.

L1 Regularization

$$\text{Cost} = \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2}_{\text{Loss function}} + \lambda \underbrace{\sum_{j=0}^M W_j^2}_{\text{Regularization Term}}$$

# L3 Regularization

L3 regularization is a type of regularization that is less commonly used, but works similarly to L2 regularization. It adds a **penalty term to the loss function that is proportional to the cube of the coefficients**. Like L2 regularization, L3 regularization leads to small coefficients and can prevent overfitting.

The choice of L1, L2, or L3 regularization will depend on the specific problem and the goals of the model. In general, **L1** regularization is useful for **feature selection**, while **L2** regularization is useful for **preventing overfitting**. L3 regularization is less commonly used and may be less effective than L1 or L2 regularization.

# Generalization

Generalization refers to the ability of a machine learning model to make accurate predictions on new, unseen data.

A model that has learned to generalize well can be used to make predictions on new data points without having seen them before.

The goal of any machine learning model is to generalize well, meaning it can make accurate predictions on new data.

The theory of generalization in machine learning refers to the mathematical framework that explains why and how machine learning models can generalize from their training data to make accurate predictions on new, unseen data.

The theory of generalization is based on the idea that the training data only provides a limited sample of the underlying distribution of the data, and that the goal of a machine learning model is to learn the underlying patterns in the data that can be used to make accurate predictions on new data.

In mathematical terms, generalization can be understood as the ability of a machine learning model to control the gap between its training error and its generalization error. The training error is the error the model makes on the training data, while the generalization error is the error the model makes on new, unseen data.

The theory of generalization is concerned with the relationship between the complexity of the model, the size of the training data, and the generalization error. It is generally accepted that as the complexity of the model increases, its ability to fit the training data improves, but its ability to generalize to new data decreases. On the other hand, as the size of the training data increases, the model's ability to generalize to new data improves, but the amount of computation required to fit the model also increases.

The goal of the theory of generalization is to understand how to choose the appropriate model complexity and training data size to achieve the best trade-off between fitting the training data and generalizing to new data.

The theory of generalization in machine learning can be mathematically formulated using the concept of capacity, the empirical risk minimization principle, and the bias-variance trade-off.

Capacity: Capacity refers to the ability of a machine learning model to fit a wide range of functions. A model with high capacity can fit a wide range of functions, while a model with low capacity can only fit a limited set of functions.

Empirical Risk Minimization: The empirical risk minimization principle states that given a set of training data, the goal of a machine learning model is to minimize the average error on the training data. This can be mathematically formulated as finding the function  $f$  that minimizes the average loss over the training data:

$$f = \operatorname{argmin}_{\{f\}} \frac{1}{m} \sum_{i=1}^m L(y_i, f(x_i))$$

where  $m$  is the number of training examples,  $(x_i, y_i)$  is the  $i$ -th training example, and  $L$  is the loss function.



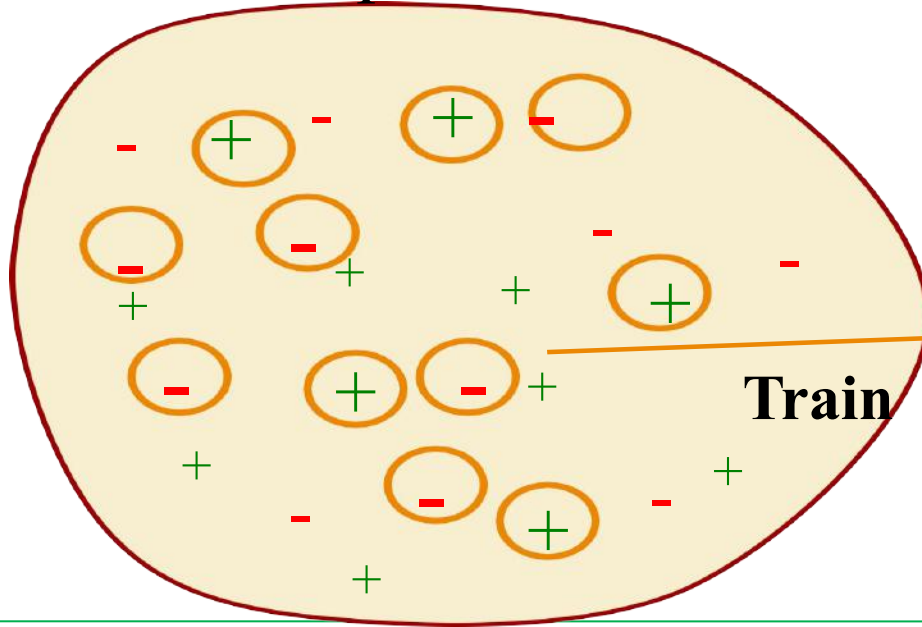
# Validation

- Cross-validation generates an approximate estimate of how well the learned model will do on “unseen” data
  - By averaging over different partitions it is more robust than just a single train/validate partition of the data
  - “k-fold” cross-validation is a generalization
    - partition data into disjoint validation subsets of size  $n/k$
    - train, validate, and average over the  $v$  partitions
    - e.g.,  $k=10$  is commonly used
- 17-08-2020
- k-fold cross-validation is approximately  $k$  times computationally more expensive than just fitting a model to all of the data
-

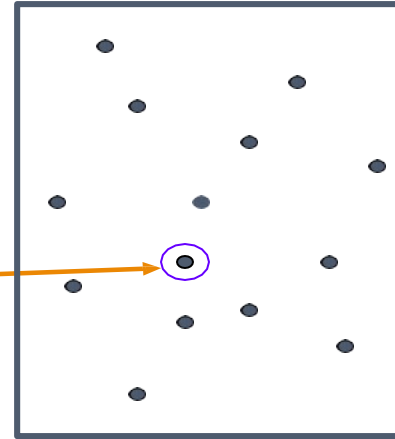
# Cross-Validation

- Split original set of examples, train

Examples  $D$



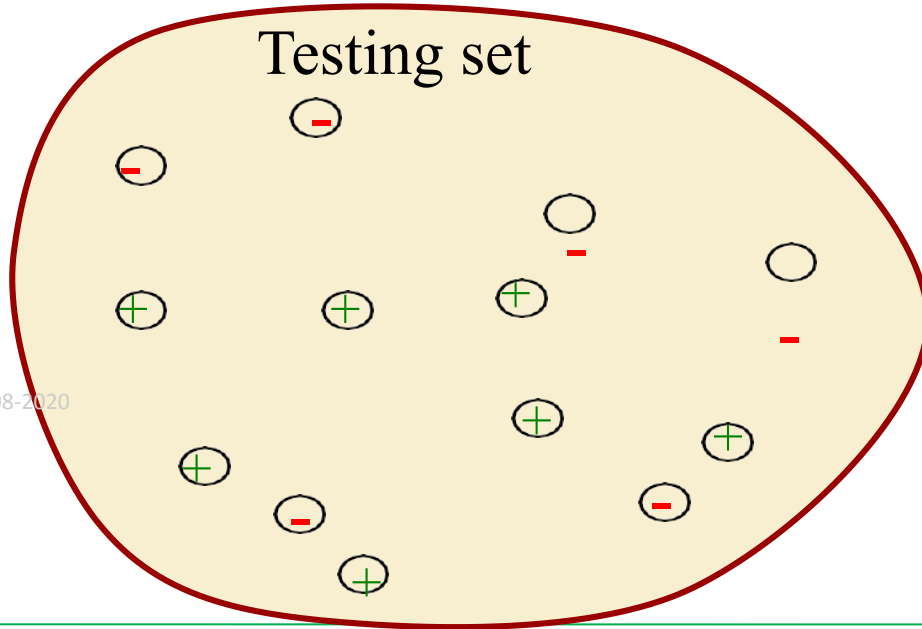
Train



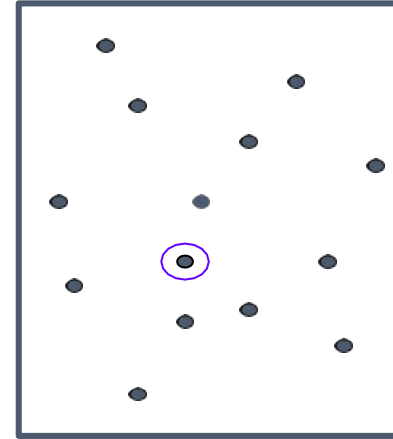
Hypothesis space  $H$

# Cross-Validation

- Evaluate hypothesis on testing set



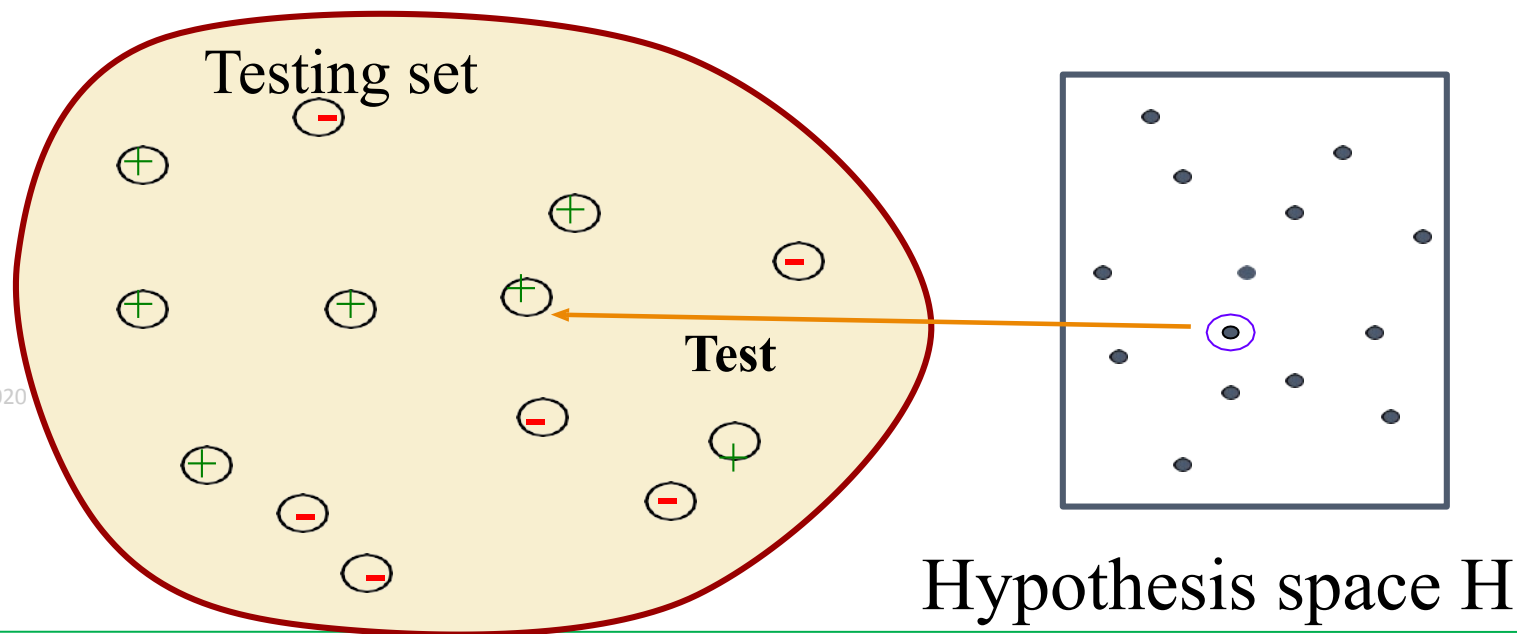
17-08-2020



Hypothesis space H

# Cross-Validation

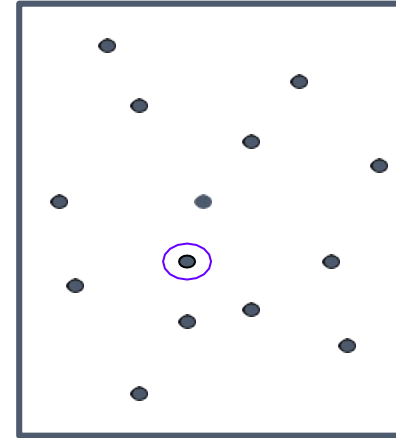
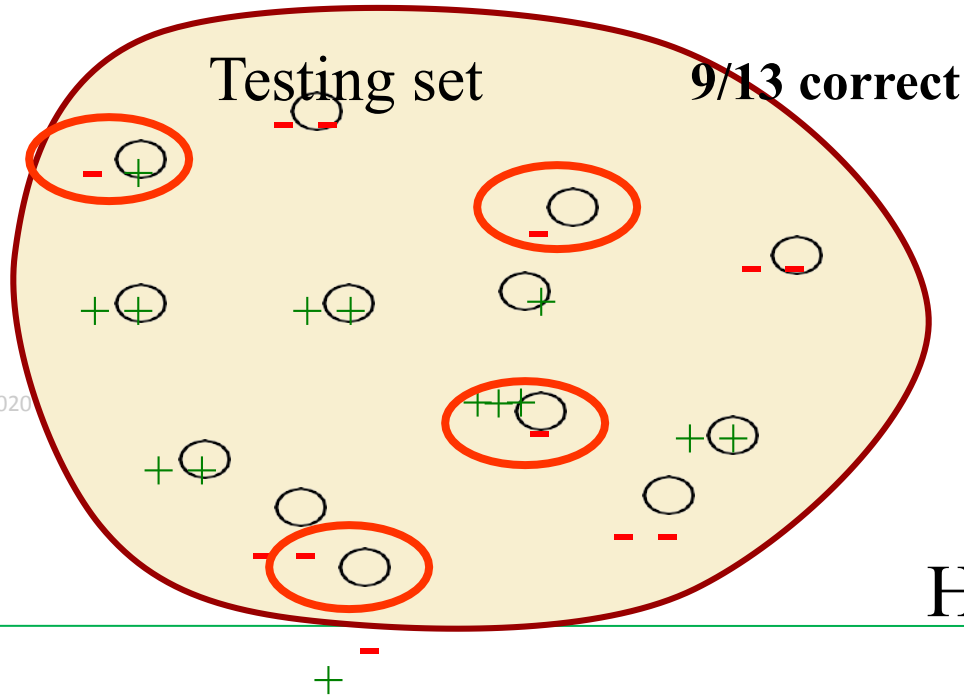
- Evaluate hypothesis on testing set



17-08-2020

# Cross-Validation

- Compare true concept against prediction



Hypothesis space H

# Common Splitting Strategies

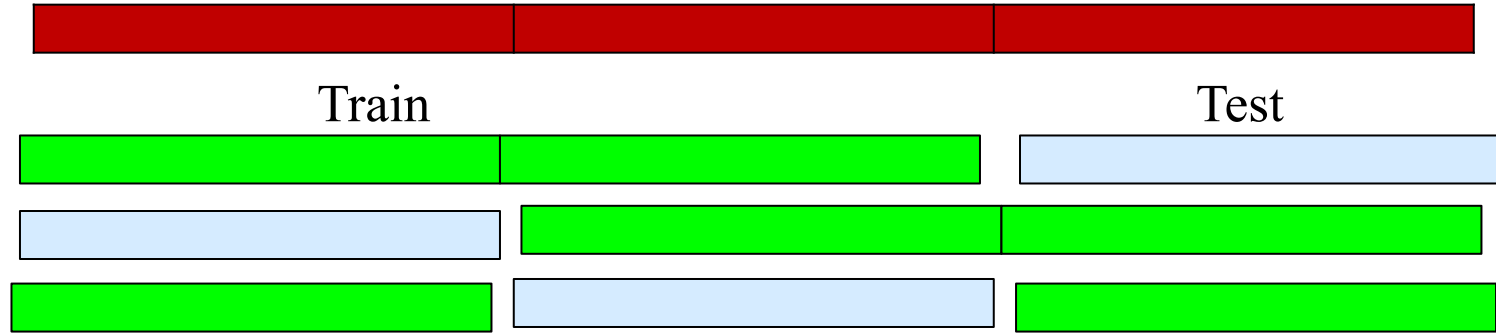
- k-fold cross-validation

Dataset

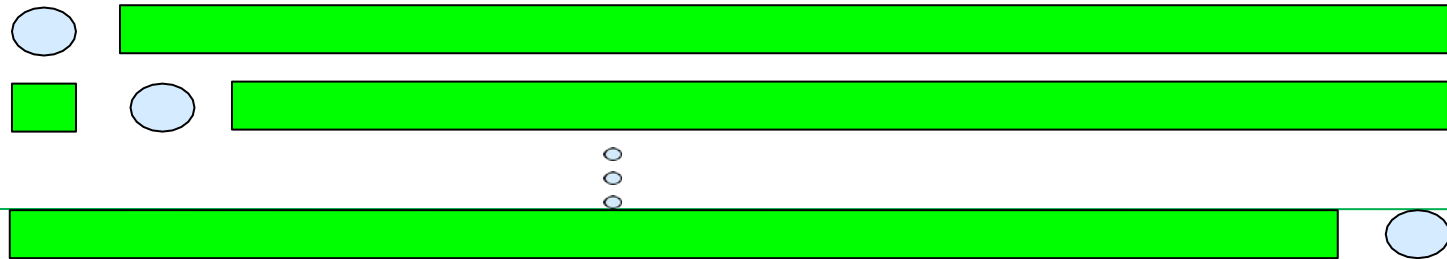


# Common Splitting Strategies

- k-fold cross-validation Dataset



- Leave-one-out (n-fold cross validation)

























# Counter Underfit

- What causes underfit?
  - Model capacity is too small to fit the training dataset as well as generalize to new dataset.
  - High bias, low variance
- Solution
  - Increase the capacity of the model
  - Examples:
    - Increase number of layers, neurons in each layer, etc.
- Result:
  - Lower Bias
  - ***Underfit* → *Good Fit*?**

# Counter Overfit

- What cause overfit?
  - Model capacity is so big that it adapts too well to training samples → Unable to generalize well to new, unseen samples
  - Low bias, high variance
- Solution
  - Regularization
- **But How?**

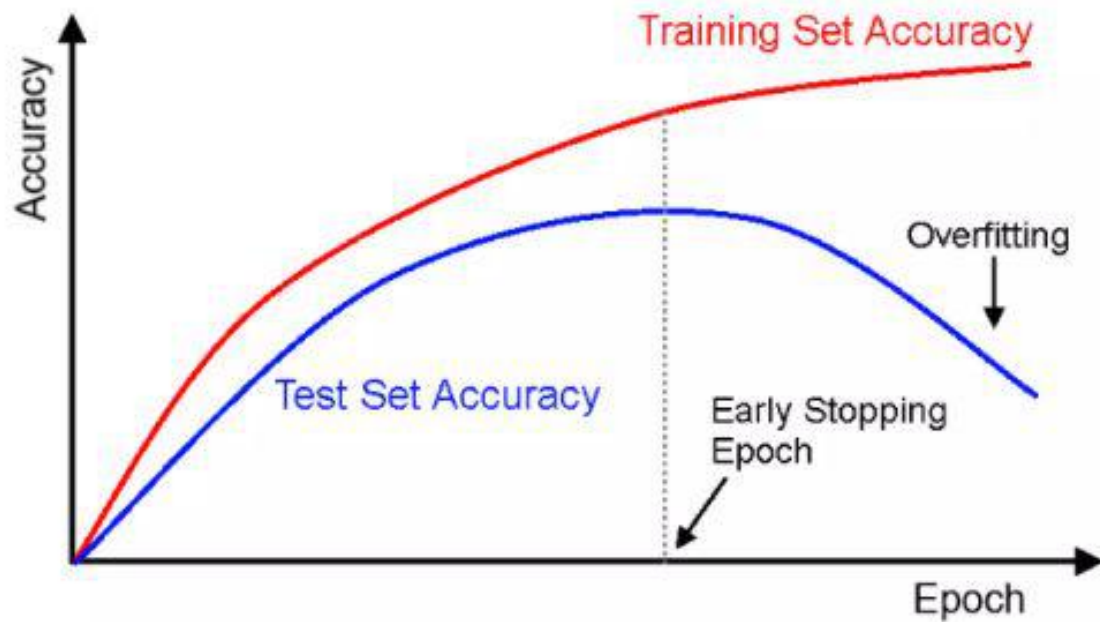
# Regularization Definition

- *Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error. [4]*

# Regularization Techniques

- **Early Stopping**
- **L1/L2**
- **Batch Norm**
- **Dropout**
- Data Augmentation
- Layer Norm
- Weight Norm

# Early Stopping

























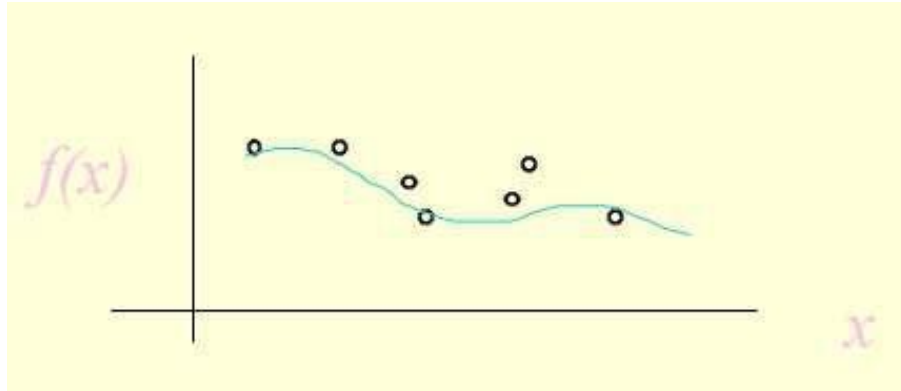
# Theory of Generalization

- In machine learning, generalization usually refers to the ability of an algorithm to be effective across a range of inputs and applications
- Our key working assumption is that data is generated by an underlying, unknown distribution  $D$ . Rather than accessing the distribution directly, statistical learning assumes that we are given a training sample  $S$ , where every element of  $S$  is i.i.d and generated according to  $D$ . A learning algorithm chooses a function (hypothesis  $h$ ) from a function space (hypothesis class)  $H$  where  $H = \{f(x, \alpha)\}$  where  $\alpha$  is the parameter vector
- We can then define the generalization error of a hypothesis  $h$  as the difference between the expectation of the error on a sample  $x$  picked from the distribution  $D$  and the empirical loss



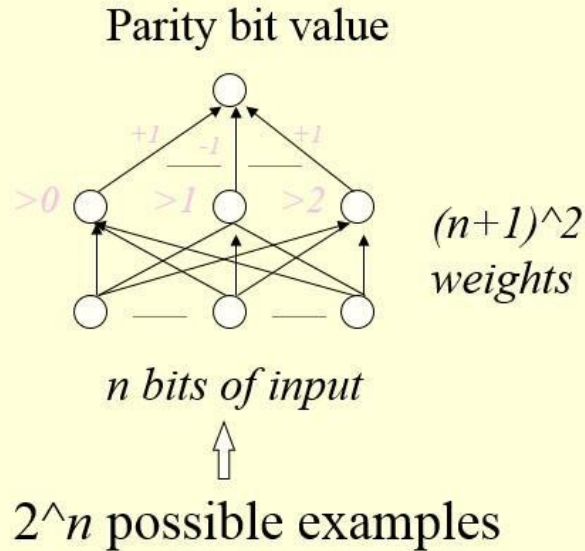
# Generalization

- The objective of learning is to achieve good generalization to new cases, otherwise just use a look-up table
- Generalization can be defined as a mathematical interpolation or regression over a set of training points:



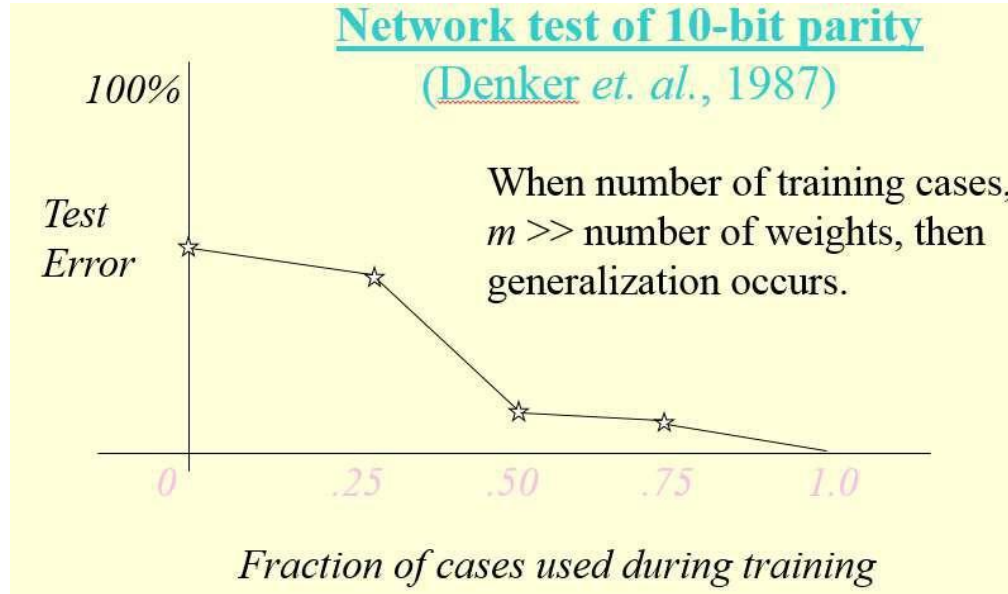
# Generalization

## An Example: Computing Parity



Can it learn from  $m$  examples to generalize to all  $2^n$  possibilities?

# Generalization



# Generalization

## A Probabilistic Guarantee

$N$  = # hidden nodes  $m$  = # training cases

$W$  = # weights  $\epsilon$  = error tolerance ( $< 1/8$ )

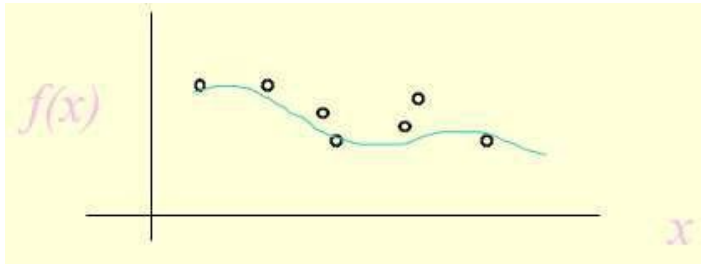
Network will generalize with 95% confidence if:

1. Error on training set  $< \epsilon/2$
2.  $m > \frac{W}{\epsilon^2} \log\left(\frac{N}{\epsilon}\right) \approx m > \frac{W}{\epsilon^2}$

Based on PAC theory  $\Rightarrow$  provides a good rule of practice.

# Generalization

- The objective of learning is to achieve good generalization to new cases, otherwise just use a look-up table
- Generalization can be defined as a mathematical interpolation or regression over a set of training points:



# Generalization

## Over-Training

- Is the equivalent of over-fitting a set of data points to a curve which is too complex
- Occam's Razor (1300s): “plurality should not be assumed without necessity”
- The simplest model which explains the majority of the data is usually the best

# Generalization

## Preventing Over-training

- Use a separate test or tuning set of examples
- Monitor error on the test set as network trains
- Stop network training just prior to over-fit error occurring- early stopping or tuning
- Number of effective weights is reduced
- Most new systems have automated early stopping methods

# Generalization

How can we control number of effective weights?

- Manually or automatically select optimum number of hidden nodes and connections
- Prevent over-fitting = over-training
- Add a weight-cost term to the bp error equation



# Generalization Bound

In order for the entire hypothesis space to have a generalization gap bigger than  $\epsilon$ , at least one of its hypothesis:  $h_1$  **or**  $h_2$  **or**  $h_3$  **or** ... etc should have. This can be expressed formally by stating that:

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |R(h) - R_{\text{emp}}(h)| > \epsilon \right] = \mathbb{P} \left[ \bigcup_{h \in \mathcal{H}} |R(h) - R_{\text{emp}}(h)| > \epsilon \right]$$

Where  $\bigcup$  denotes the union of the events, which also corresponds to the logical **OR** operator. Using the union bound inequality, we get:

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |R(h) - R_{\text{emp}}(h)| > \epsilon \right] \leq \sum_{h \in \mathcal{H}} \mathbb{P}[|R(h) - R_{\text{emp}}(h)| > \epsilon]$$

We exactly know the bound on the probability under the summation from our analysis using the Hoeffding's inequality, so we end up with:

# Generalization Bound

$$\mathbb{P} \left[ \sup_{h \in \mathcal{H}} |R(h) - R_{\text{emp}}(h)| > \epsilon \right] \leq 2|\mathcal{H}| \exp(-2m\epsilon^2)$$

Where  $|\mathcal{H}|$  is the size of the hypothesis space. By denoting the right hand side of the above inequality by  $\delta$ , we can say that with a confidence  $1 - \delta$ :

$$|R(h) - R_{\text{emp}}(h)| \leq \epsilon \Rightarrow R(h) \leq R_{\text{emp}}(h) + \epsilon$$

And with some basic algebra, we can express  $\epsilon$  in terms of  $\delta$  and get:

$$R(h) \leq R_{\text{emp}}(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln \frac{2}{\delta}}{2m}}$$

# Generalization Bound

- There are two types of bound
  - VC generalization bound
  - Distributed function based bound

## VC generalization bound

$$R(h) \lesssim \hat{R}_n(h) + \epsilon(\mathcal{H}, n)$$

# Approximation- Generalization Tradeoff

Given a set  $\mathcal{H}$ , find a function  $h \in \mathcal{H}$  that minimizes  $R(h)$

Our goal is to find an  $h \in \mathcal{H}$  that approximates the Bayes classifier, or some true underlying function

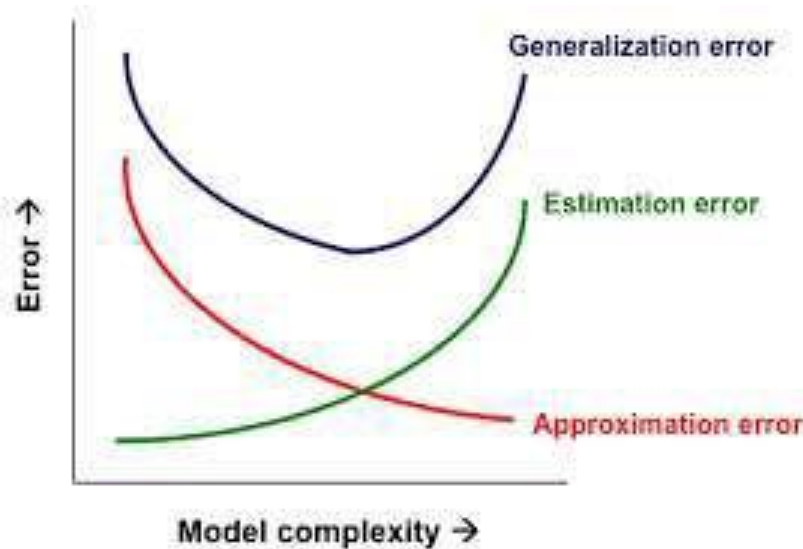
More complex  $\mathcal{H}$   $\Rightarrow$  better chance of **approximating** the ideal classifier/function

Less complex  $\mathcal{H}$   $\Rightarrow$  better chance of **generalizing** to new data (out of sample)

Regularization plays a similar role, by biasing us away from complex classifiers/functions

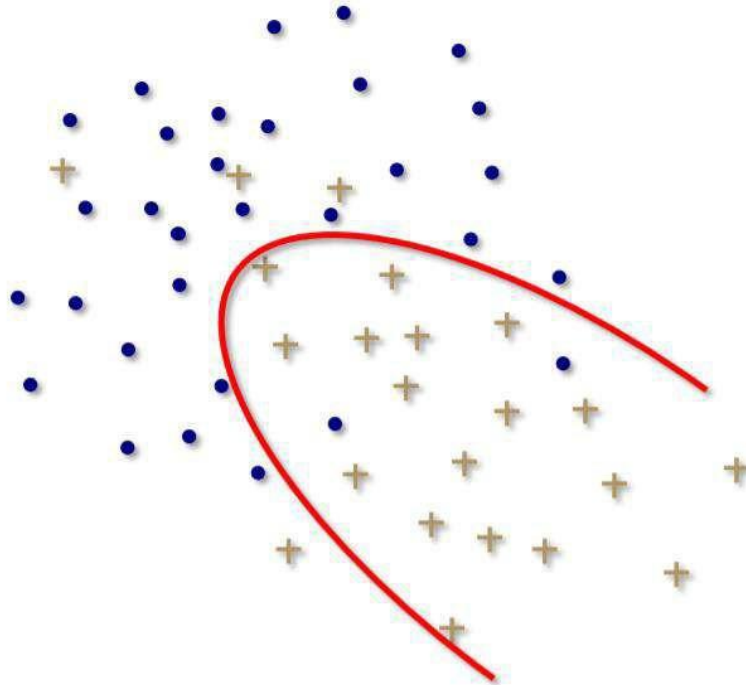
We must carefully limit “complexity” to avoid **overfitting**

# Approximation- Generalization Tradeoff

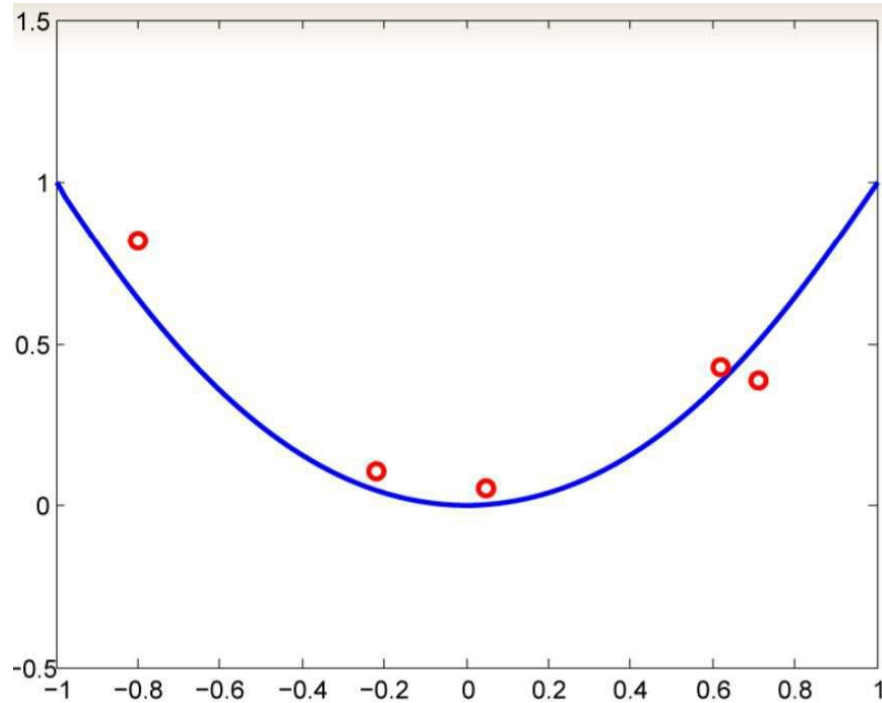


# Overfitting

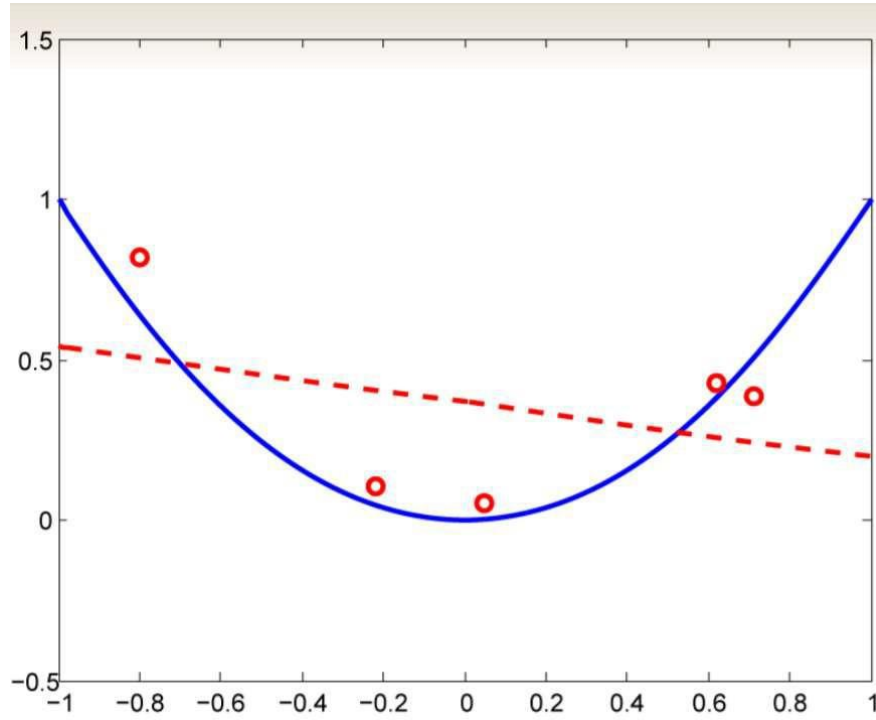
---



# Overfitting

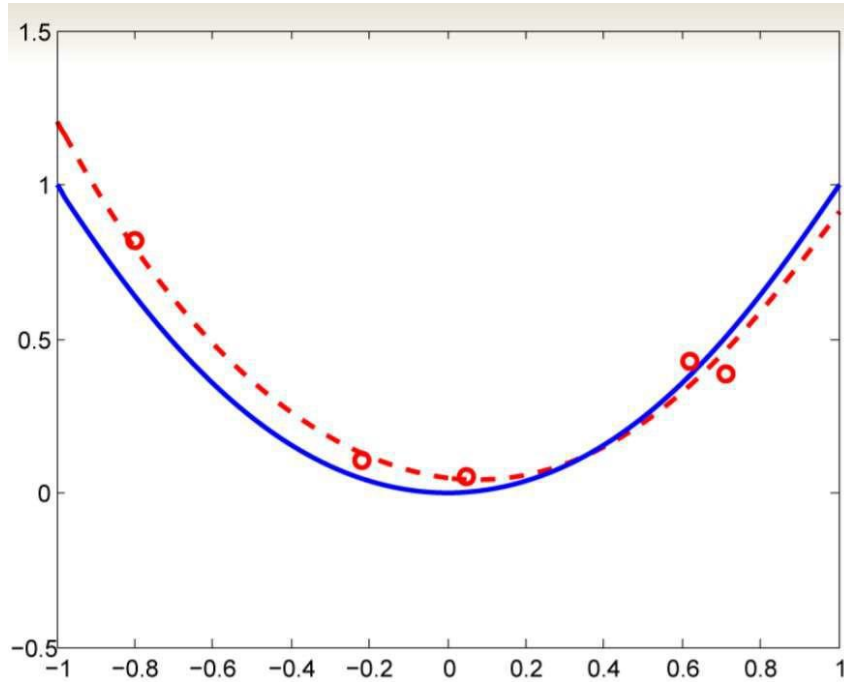


# Overfitting

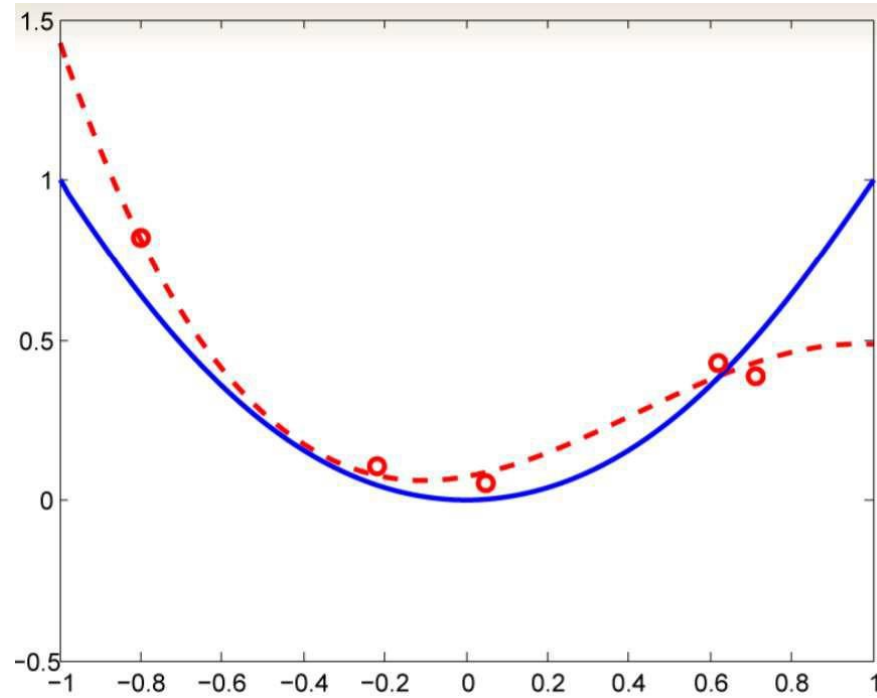




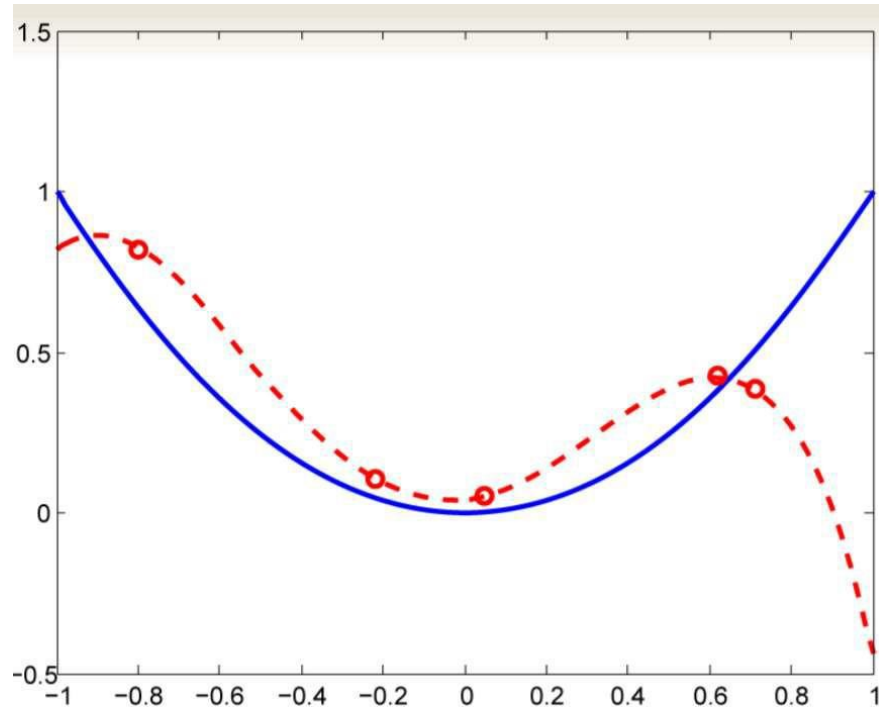
# Overfitting



# Overfitting



# Overfitting



## Bias and Variance

Bias is the difference between the Predicted Value and the Expected Value.

Mathematically, let the input variables be  $X$  and a target variable  $Y$ . We map the relationship between the two using a function  $f$ . Therefore,

$$Y = f(X) + e$$

Here 'e' is the error that is normally distributed. The aim of our model  $f(x)$  is to predict values as close to  $f(x)$  as possible. Here, the Bias of the model is:

$$\text{Bias}[f(X)] = E[f(X) - f(X)]$$

As I explained above, when the model makes the generalizations i.e. when there is a high bias error, it results in a very simplistic model that does not consider the variations very well. Since it does not learn the training data very well, it is called Underfitting.

## Variance

Contrary to bias, the Variance is when the model takes into account the fluctuations in the data i.e. the noise as well. So, what happens when our model has a high variance?

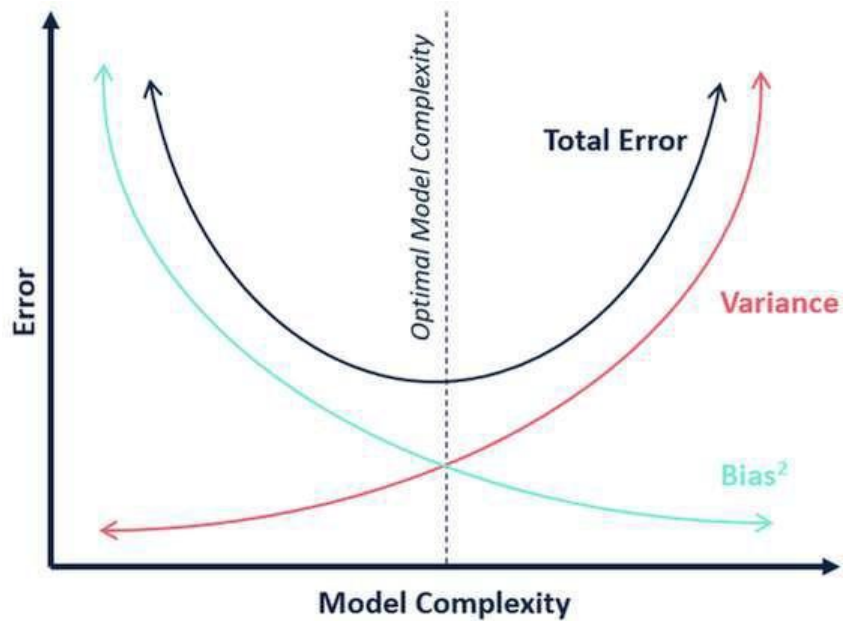
The model will still consider the variance as something to learn from. That is, the model learns too much from the training data, so much so, that when confronted with new (testing) data, it is unable to predict accurately based on it.

Mathematically, the variance error in the model is:

$$\text{Variance}[f(x)] = E[X^2] - E[X]^2$$

Since in the case of high variance, the model learns too much from the training data, it is called **overfitting**.

# Bias and Variance



# Learning curves

- Learning curves are plots that show changes in learning performance over time in terms of experience
- Learning curves of model performance on the train and validation datasets can be used to diagnose an underfit, overfit, or well-fit model
- Learning curves of model performance can be used to diagnose whether the train or validation datasets are not relatively representative of the problem domain
- Generally, a learning curve is a plot that shows time or experience on the x-axis and learning or improvement on the y-axis

Learning curves are deemed effective tools for monitoring the performance of workers exposed to a new task. LCs provide a mathematical representation of the learning process that takes place as task repetition occurs.

# Learning curves

- **Train Learning Curve:** Learning curve calculated from the training dataset that gives an idea of how well the model is learning
- **Validation Learning Curve:** Learning curve calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing
- **Optimization Learning Curves:** Learning curves calculated on the metric by which the parameters of the model are being optimized, e.g. loss
- **Performance Learning Curves:** Learning curves calculated on the metric by which the model will be evaluated and selected, e.g. accuracy

There are three common **dynamics** that you are likely to observe in learning curves; they are:

- Underfit
- Overfit
- Good Fit



## Learning curves

- Underfitting refers to a model that cannot learn the training dataset.
- A plot of learning curves shows underfitting if:
  - The training loss remains flat regardless of training
  - The training loss continues to decrease until the end of training
- Overfitting refers to a model that has learned the training dataset too well, including the statistical noise or random fluctuations in the training dataset.
- A plot of learning curves shows overfitting if:
  - The plot of training loss continues to decrease with experience
  - The plot of validation loss decreases to a point and begins increasing again

## Learning curves

- A good fit is the goal of the learning algorithm and exists between an overfit and underfit model
- A good fit is identified by a training and validation loss that decreases to a point of stability with a minimal gap between the two final loss values
- A plot of learning curves shows a good fit if:
  - The plot of training loss decreases to a point of stability
  - The plot of validation loss decreases to a point of stability and has a small gap with the training loss