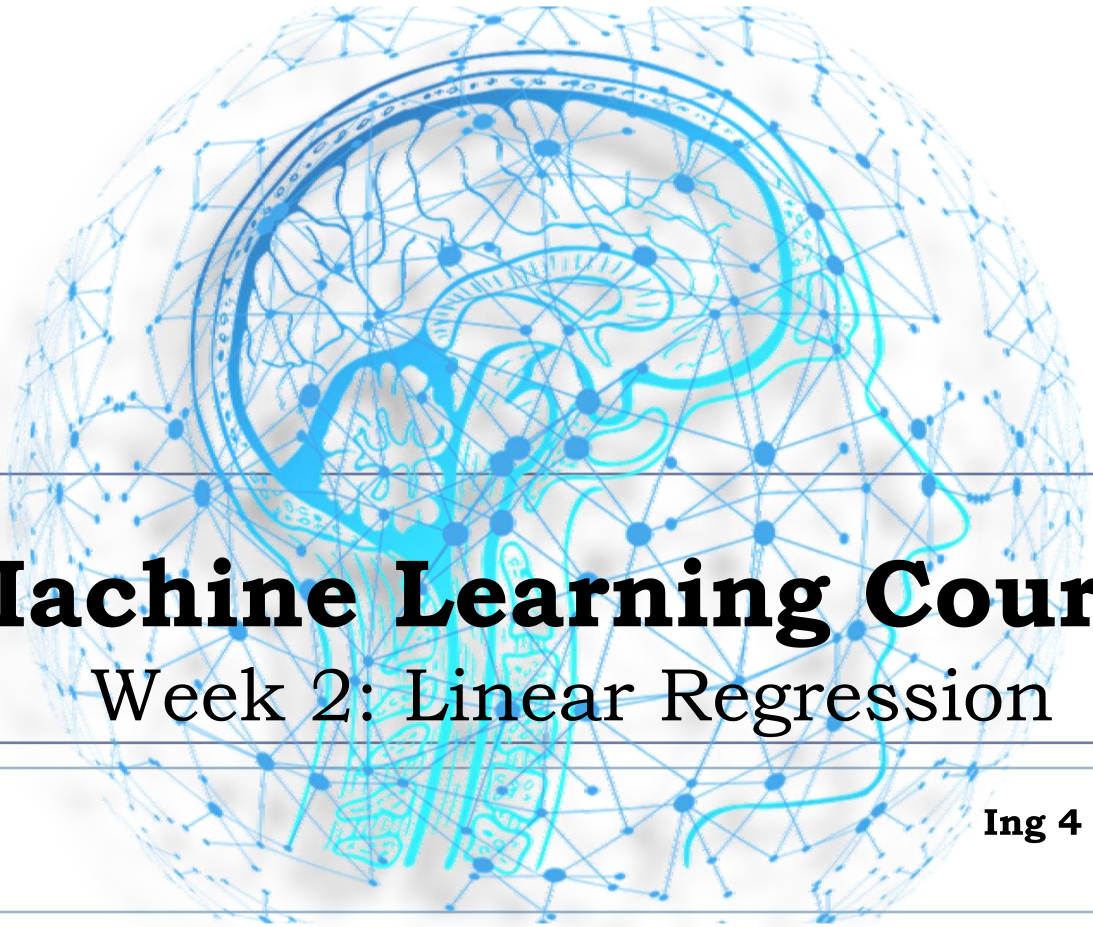


Machine Learning Course

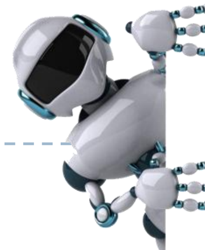
Week 2: Linear Regression

Ing 4 SI, 2022

Pr. Khadija SLIMANI

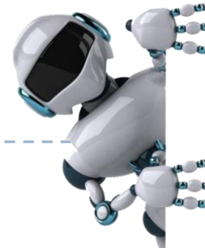
Course overview …..

1. Week 1 : Introduction to Data Science and Machine Learning
2. **Week 2: Univariate & Multivariate Linear Regression**
3. Week 3: Logistic Regression (Classification)
4. Week 4: Decision Trees (Regression & Classification)
5. Week 5: Model evaluation (overfitting, bias-variance, crossfolding, ...)



Course overview

1. Week 2 : Univariate & Multivariate Linear Regression
    1. Introduction to Linear Regression
    2. Simple Linear Regression
    3. Multiple Linear Regression
    4. Evaluation of a Linear Regression Model
    5. Practical Work

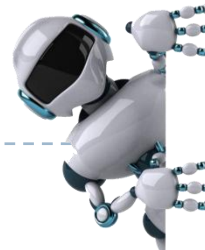


1.1
What is Linear Regression ?

What is Linear Regression ?

- **Linear Regression** is a statistical model used to predict the relationship between independent and dependent variables.

- In **regression** analysis, the **dependent variable** is denoted "Y" and the **independent** variables are denoted by "X".



What is Linear Regression ?

- **Goal**: predict real (continuous) valued outputs, by modeling how our observations that are associated with some features change as we change the values of theses features.
- Example: training set of housing prices

X= input, features, covariate, predictors

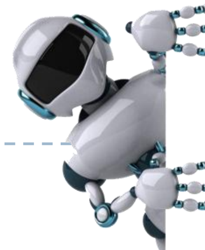
y= output, target

m= number of training examples

	Size (m2)	Built Year	Nb bathrooms	......	Sale price (k$)
$x^{(1)}$	200	2010	2	....	$y^{(1)} = 800$
$x^{(2)}$	300	1995	2	.....	$y^{(2)} = 750$
....	......	......	......	.....	......

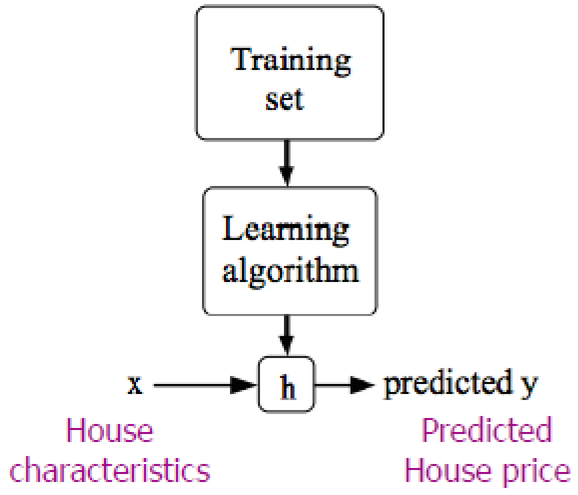
- Regression is about learning the relationship between X and y, and using it to predict the house price of new data

250	2005	3	.....	????????

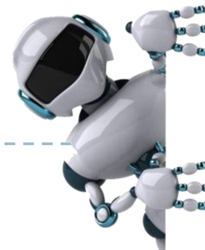


Model Representation

- X(i) denotes the “input” variables (house characteristics)
- Y(i) denotes the “output” or target variable that we are trying to predict (price)
- A pair(x(i), y(i)) is called a training example
- A list of m training examples(x(i), y(i)); i=1,...,m—is called a training set

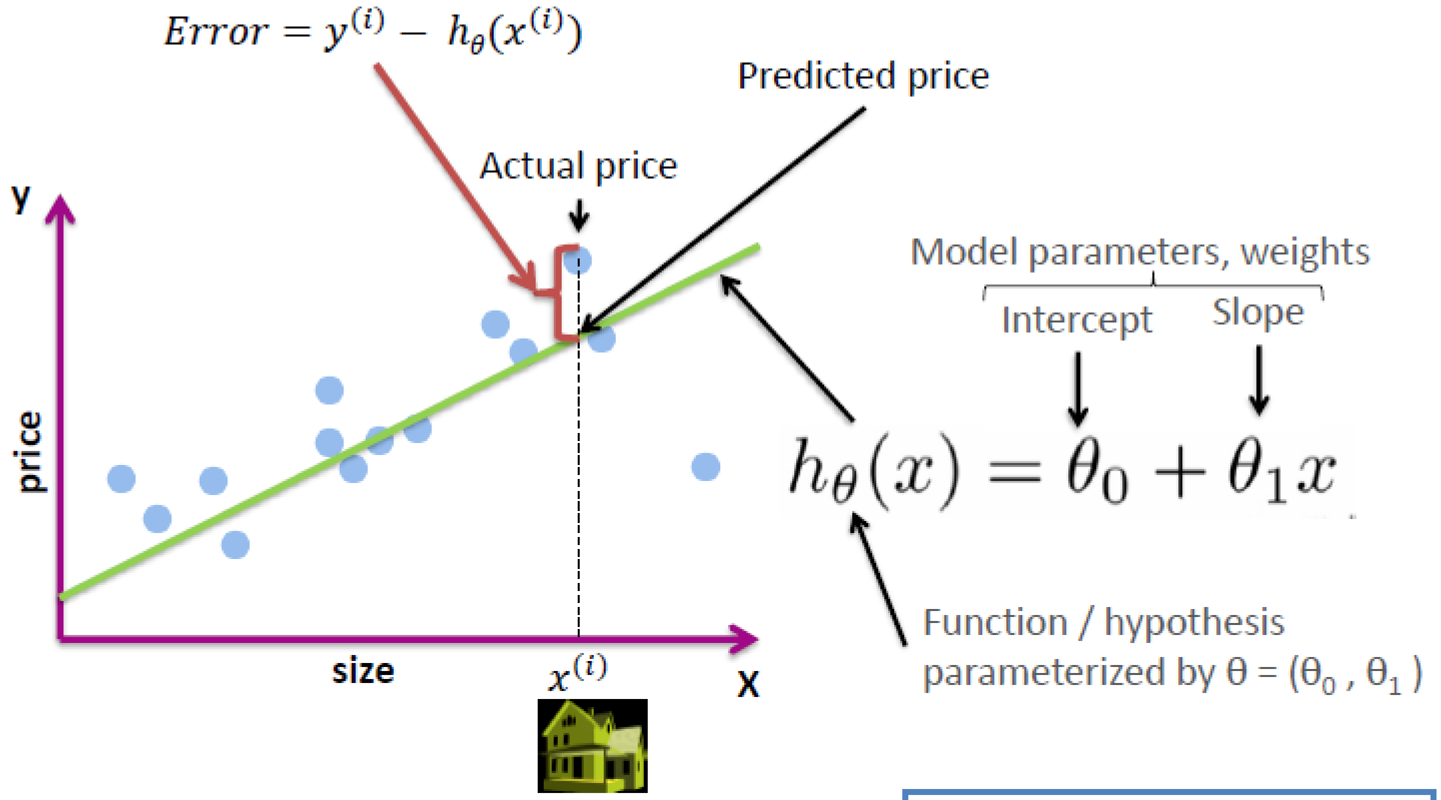


h: X → Y
Hypothesis or function that takes as input the house’s characteristics to estimate its price.



1.2: Simple/Univariate Linear Regression

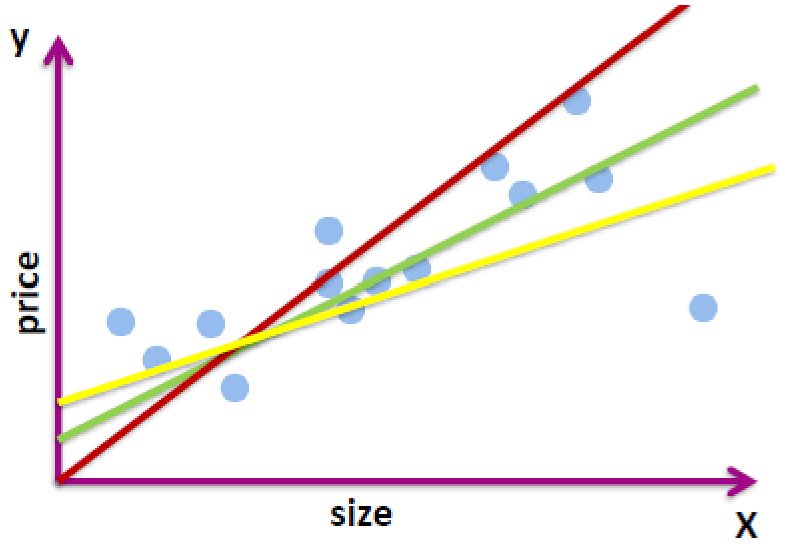
Simple Linear Regression: Model



$Error = y^{(i)} - h_\theta(x^{(i)})$
Predicted price
Actual price
Model parameters, weights
Intercept
Slope
$h_\theta(x) = \theta_0 + \theta_1 x$
Function / hypothesis parameterized by θ = (θ_0 , θ_1)
y
price
size
$x^{(i)}$
X

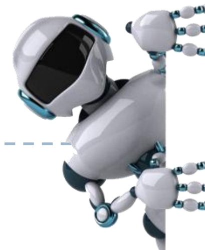
$$\boxed{y = \theta_0 + \theta_1 x + \varepsilon}$$

Model Evaluation



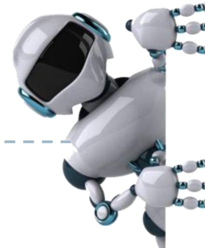
Each line has different parameters $\theta = (\theta_0 , \theta_1)$
➔ different errors

- Which line is the best fit ?
- **What should a good function hθ(x) minimize?**
  - Sum error on all data points
  - Sum abs(error) on all data points
  - Sum error^2 on all data points



Model Evaluation : Cost function

- The cost function is the **technique of evaluating "the performance of our algorithm/model"**.
- It takes both predicted outputs by the model and actual outputs and calculates how much wrong the model was in its prediction.
- It outputs a higher number if our predictions differ a lot from the actual values.



Model Evaluation : Cost function

- The best hypothesis $h_\theta(x)$is the one that minimizes the cost function

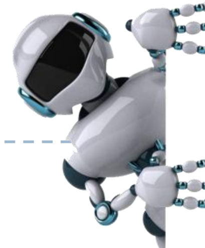
Sum over all data points

Actual

$$J(\theta_0, \theta_1) = 1/\mathrm{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

Predicted

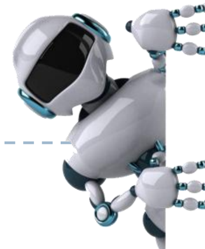
1/m - means we determine the average
1/2m, the 2 makes the math a bit easier, and doesn't change the weights θ we determine at all (i.e. half the smallest value is still the smallest value!)



Model Evaluation : Cost function

- In Machine Learning, our main goal is to minimize the error which is defined by the Loss Function. And every type of Algorithm has different ways of measuring the error.

- Cost function and loss function are synonymous and used interchangeably but they are “different”.
  - A loss function/error function is for a single training example/input.
  - A cost function, on the other hand, is the average loss over the entire training dataset.



Model Evaluation : Sum of Errors (SE)

- Some basic Loss Functions used in Regression Algorithms:

  - **Sum of Errors (SE):** the most basic loss function which is nothing but the sum of errors in each iteration. The error will be the difference in the predicted value and the actual value. So the loss function will be given as:

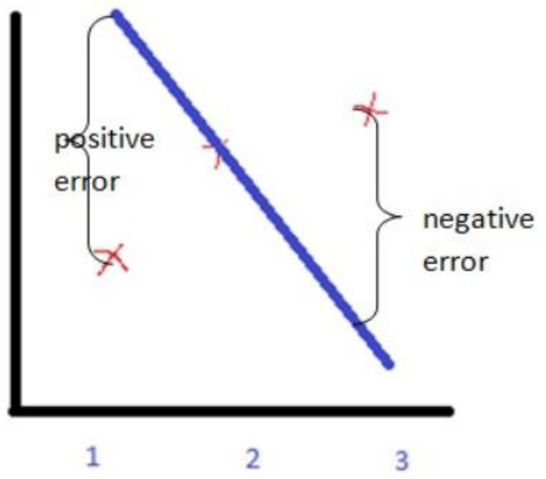
$$\mathbf{L} = \Sigma(\hat{Y} - Y)$$

$\hat{Y}$ is the predicted value; Y is the actual value

  - *where the summation goes from n=1 to N where N is the number of instances in our dataset.*

Model Evaluation : Sum of Errors (SE)

- Now consider the following line fitting our 3 data points:
  - For point 3 the error is negative as the predicted value is lower. Whereas for point 1, the error is positive and of almost the same magnitude. For point 2 it is 0. Adding all of these up would lead to a total error of 0! But the error is certainly much more than that.
  - If the error is 0 then the **algorithm will assume that it has converged** when it actually hasn't *and will exit prematurely*. It would show a very less error value where in reality the value would be much larger.
  - So how can you claim that this is the wrong line? You actually cannot. You just chose the wrong loss function.



Model Evaluation : Sum of Absolute Errors (SAE)

- SE was certainly not the loss function we'd want to use. So let's change it a bit to overcome its shortcoming. Let's just take the absolute values of the errors for all iterations. This should solve the problem.. right? Or no?

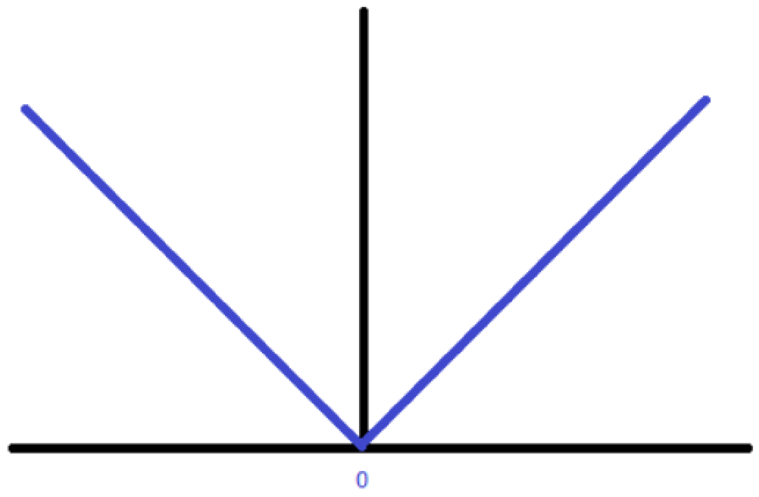
- This is how the loss function would look like:

$$\mathbf{L} = \Sigma(|\hat{Y} - Y|)$$

- So now the error terms won't cancel out each other and will actually add up. **So any potential problem with this function?**

Model Evaluation : Sum of Absolute Errors (SAE)

- Well, yes. This loss function is not differentiable at 0. The graph of the loss function will be:



Y axis is the loss function

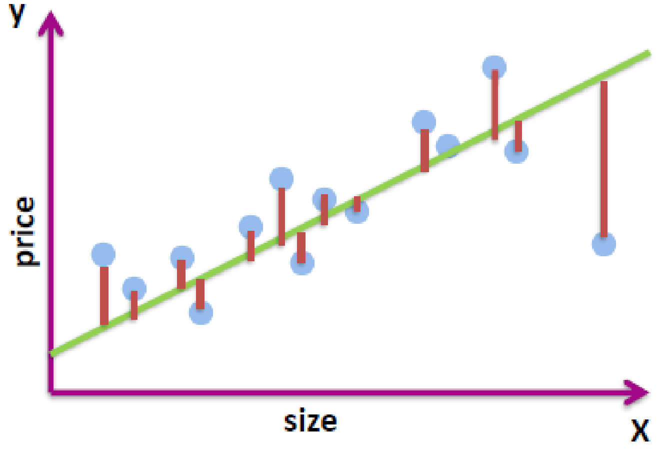
- The derivative *will not exist* at 0. We need to differentiate the function and equate it to 0 to find the optimum point. And that won't be possible here. We won't be able to solve for the solution.

Model Evaluation : Sum of Squared Errors(SSE)

- So let's take the squares instead of the absolutes. The Sum of Squared Errors (SSE) is also called Residual Sum of Squares (RSS). The loss function will now become:

$$\mathbf{L} = [\,\Sigma(\hat{Y} - Y)^2]$$

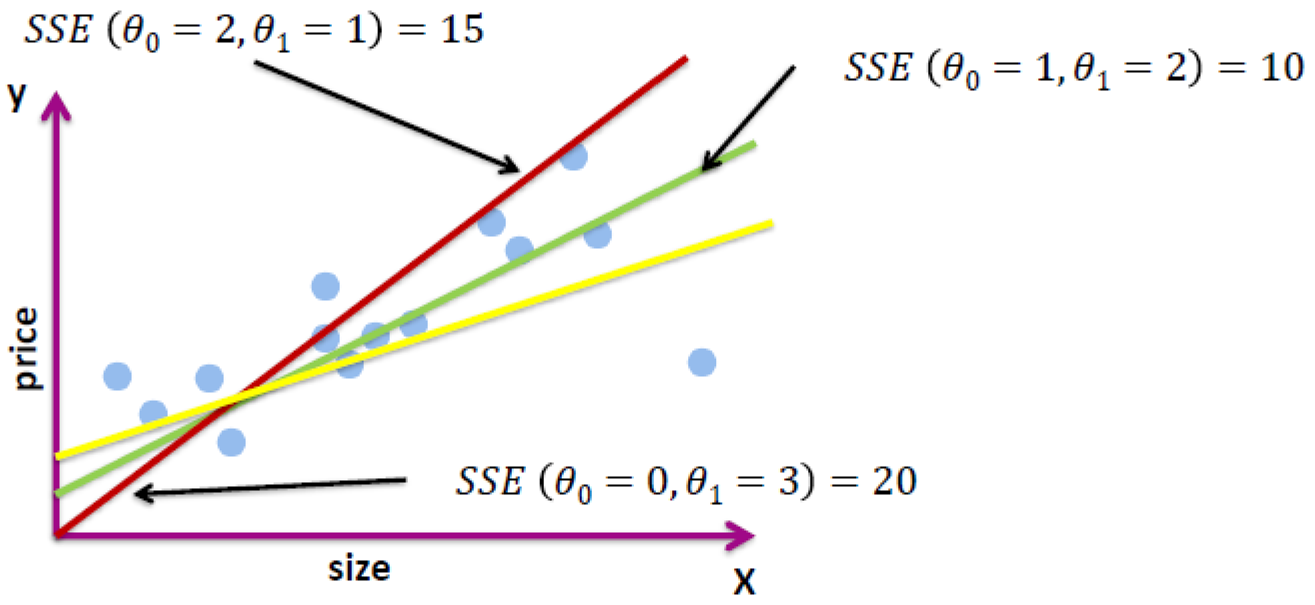
- which is very much differentiable at all points and gives non-negative errors.



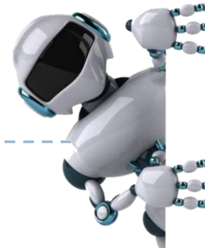
$$SSE\,(\theta_0, \theta_1) =$$

$$\begin{aligned}
&(\,y^{(1)} - (\theta_0 + \theta_1 * x^{(1)})\,)^2 + \\
&(\,y^{(2)} - (\theta_0 + \theta_1 * x^{(2)})\,)^2 + \\
&\ldots\ldots\ldots\ldots\ldots\ldots + \\
&(\,y^{(m)} - (\theta_0 + \theta_1 * x^{(m)})\,)^2
\end{aligned}$$

Model Evaluation : Sum of Squared Errors(SSE)

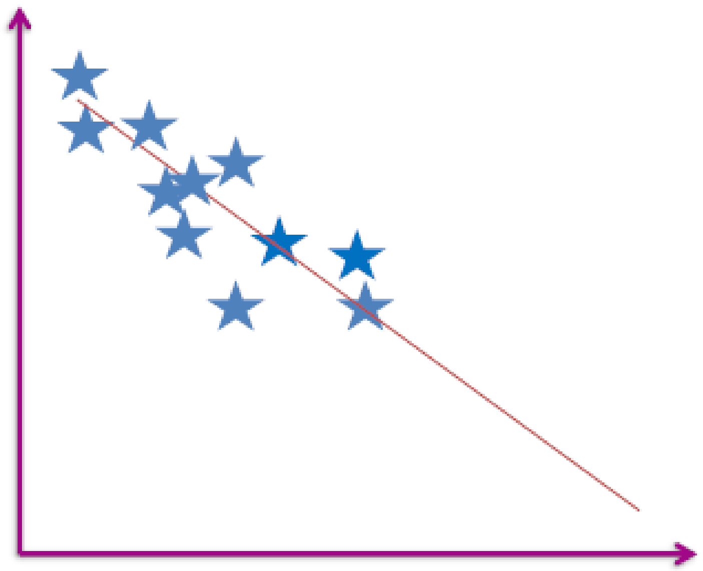


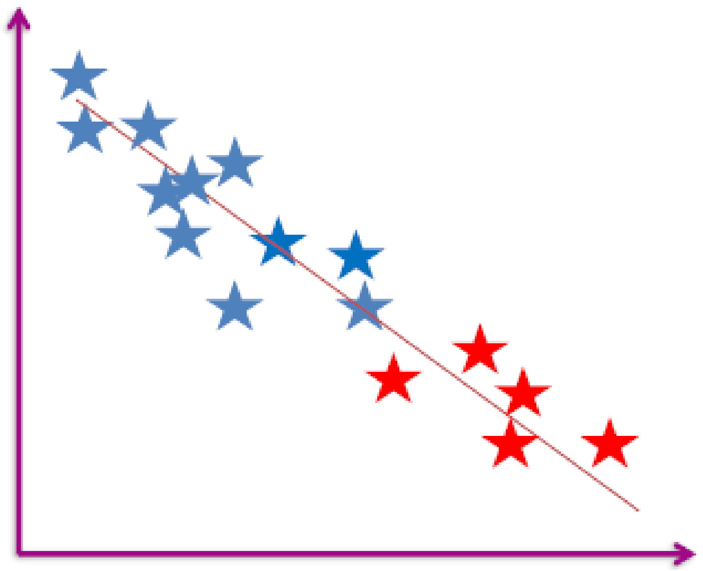
- The green line is a better fit.



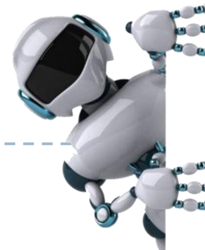
Model Evaluation : Sum of Squared Errors(SSE)

- Which fit has larger SSE ?





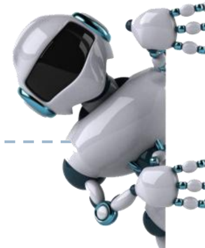
- Larger SSE doesn't necessarily mean worst fit →Need an other metric.



Model Evaluation : R Squared (R2)

- R2 score is a metric that answers the questions: "*how much of variability in the output (y) is explained by the change in the input (x)*".
- In contrast, MAE and MSE depend on the context as we have seen whereas the R2 score is independent of context.

- So, with help of R squared we have a baseline model to compare a model which none of the other metrics provides. So basically R2 calculates how must regression line is better than a mean line.
- Hence, R squared is also known as Coefficient of Determination or sometimes also known as Goodness of fit.

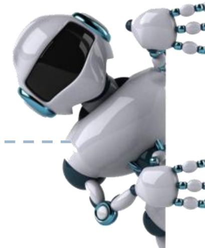


Model Evaluation : R Squared (R2)

- To calculate R2, we use the formula:

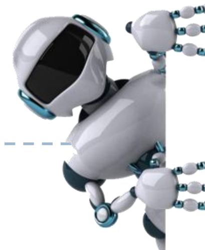
$$R^2 = \frac{TSS - SSE}{TSS} = 1 - \frac{SSE}{TSS} \qquad TSS = \sum_{i=1}^{m} (y^{(i)} - \bar{y})^2$$

- An R2 statistic that is close to 1 indicates that a large proportion of the variability in the response has been explained by the regression.
- A number near 0 indicates that the regression did not explain much of the variability in the response.



Model Optimization

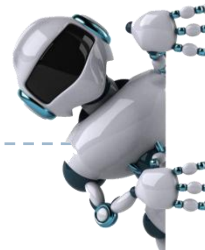
- **Let’s see how to find the best line automatically.**
- The learning algorithm should find $\theta^* = (\theta_0^*, \theta_1^*)$ that minimizes this cost.
- There are many different methods that we can apply to our linear regression model in order to make it more efficient. The most common of them are :
  - Gradient Descent
  - Ordinary least square (OLS): Used in the linear regression in python (sklearn)
  - Least Square Method / Normal Equation Method
  - Adams Method
  - Singular Value Decomposition (SVD)



Model Optimization: Gradient Descent

- **Gradient descent** is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient.

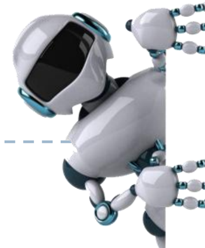
- In machine learning, we use gradient descent to update the parameters of our model. Parameters refer to **coefficients** in Linear Regression and **weights** in neural networks.



Model Optimization: Gradient Descent

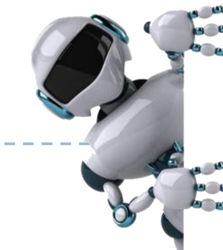
- Let's suppose we have our data plotted out in the form of a scatter graph, and when we apply a cost function to it, our model will make a prediction. Now this prediction can be very good, or it can be far away from our ideal prediction (meaning its cost will be high). So, in order to minimize that cost (error), we apply gradient descent to it.

- Now, gradient descent will slowly converge our hypothesis towards a global minimum, where the cost would be lowest.



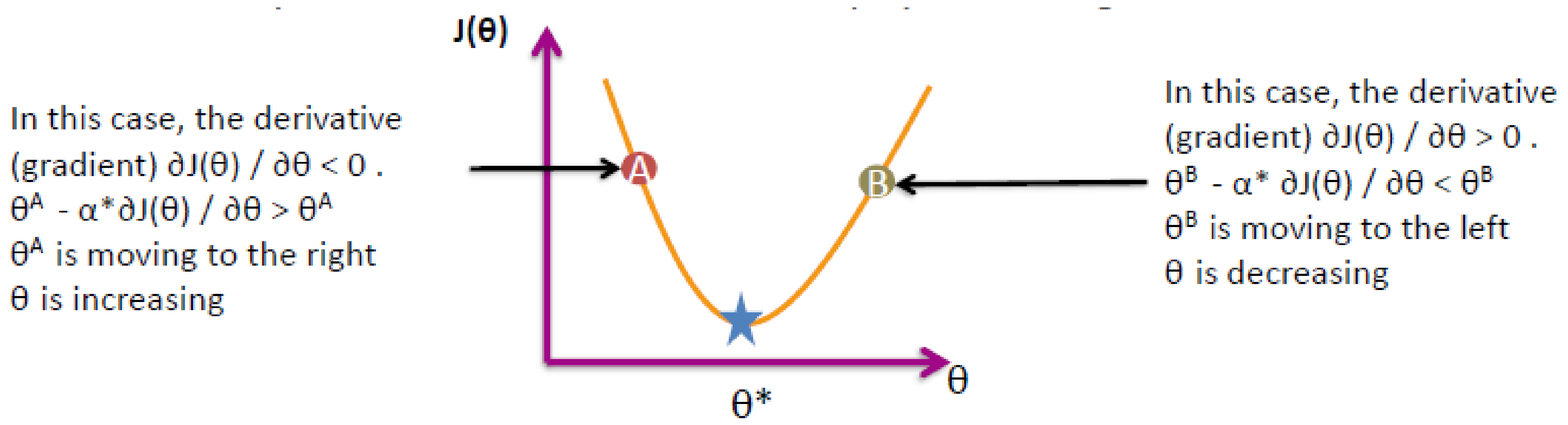
Model Optimization: Gradient Descent

- In doing so, we have to manually set the value of alpha, and the slope of the hypothesis changes with respect to our alpha’s value. If the value of alpha is large, then it will take big steps. Otherwise, in the case of small alpha, our hypothesis would converge slowly and through small baby steps.

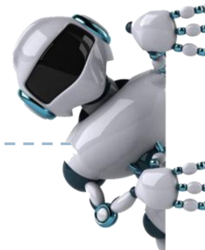


Model Optimization: Gradient Descent

- GD starts by a random initial θ and iteratively update it to get towards θ*.


J(θ)
In this case, the derivative (gradient) ∂J(θ) / ∂θ < 0 .
θ^A - α*∂J(θ) / ∂θ > θ^A
θ^A is moving to the right
θ is increasing
A
B
In this case, the derivative (gradient) ∂J(θ) / ∂θ > 0 .
θ^B - α* ∂J(θ) / ∂θ < θ^B
θ^B is moving to the left
θ is decreasing
θ*
θ

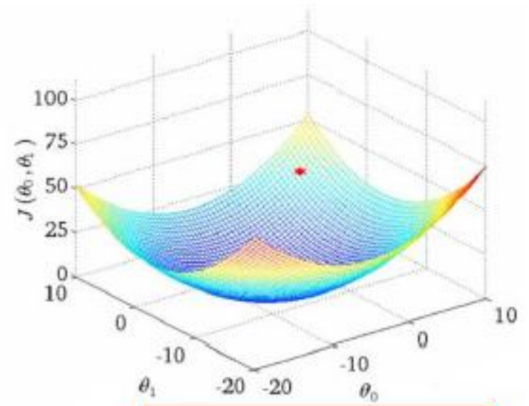
- α is called the learning rate or the step size
  - **Too small:** Take baby steps -> Take too long to converge.
  - **Too large:** Can overshoot the minimum -> fail to converge.



Simple Linear Regression with GD

- Repeat until convergence :

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \qquad (\text{for } j = 0 \text{ and } j = 1)$$

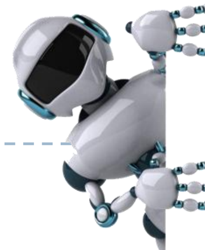


$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x}$$

- If we calculate the derivatives, the expression becomes:

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \qquad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

- Convergence means either:
  - The cost function is no longer changing by more than ε.
  - The number of iterations is reached



Ordinary Least Square Algorithm

- The Ordinary Least Square (OLS) approach chooses θ₀, θ₁ to minimize SSE.

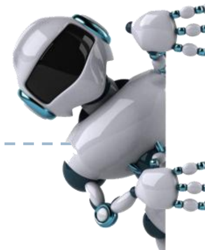
$$SSE = \sum_{i=1}^{m} (y^{(i)} - \theta_0 - \theta_1 * x^{(i)})^2$$

- In this method, we will minimize SSE by explicitly taking its derivatives with respect to the θ₀, θ₁, and setting them to zero.
- The minimizing values can be shown to be:

$$\widehat{\theta_1} = \frac{\sum_{i=1}^{m} (x^{(i)} - \bar{x}) (y^{(i)} - \bar{y})}{\sum_{i=1}^{m} (x^{(i)} - \bar{x})^2}$$

$$\widehat{\theta_0} = \bar{y} - \widehat{\theta_1} * \bar{x}$$

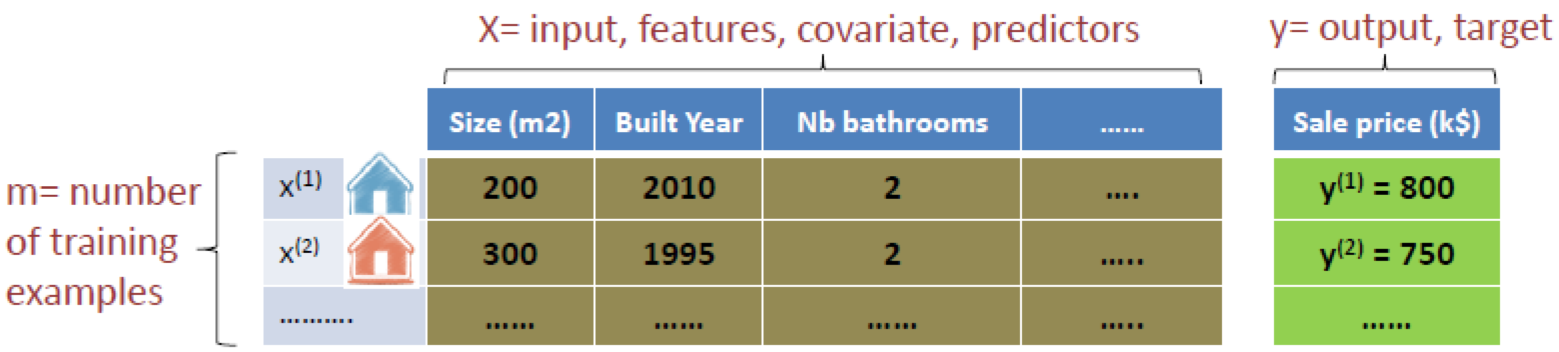
- Where $\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y^{(i)}$, $\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$ are the sample means.



1.3
Multivariate Linear Regression

Multiple Linear Regression

- In simple linear regression, we use one feature x to predict (y)
- In multiple linear regression, we have multiple features $X=(x_1,x_2, \dots, x_n)$



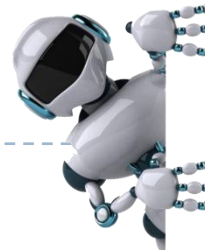
X= input, features, covariate, predictors

y= output, target

m= number of training examples

	Size (m2)	Built Year	Nb bathrooms	……	Sale price (k$)
$x^{(1)}$	200	2010	2	….	$y^{(1)}$ = 800
$x^{(2)}$	300	1995	2	…..	$y^{(2)}$ = 750
……….	……	……	……	…..	……

- $X_{(i)}$ is an n-dimensional feature vector
- $X_{(1)} =(200, 2010, 2, \dots)^T$ the feature vector of the first training example.
- $x_{j(i)}$ is the value of feature j in the ith training example. $x_{2(1)}= 2010$



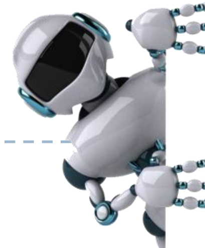
Multiple Linear Regression

- In simple linear regression, $h_\theta(x) = \theta_0 + \theta_1 x$
- In multiple linear regression, $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
- For convenience of notation, define $x_0 = 1$ $(x_0^{(i)} = 1)$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

- Rewrite in matrix notation

$$\begin{matrix}\theta_0\\ \theta_1\\ \theta_2\\ \ldots\\ \theta_n\end{matrix} \qquad \theta, x \in R^{n+1} \qquad \begin{matrix}x_0\\ x_1\\ x_2\\ \ldots\\ x_n\end{matrix} \qquad h_\theta(x) = \sum_{i=1}^{n} \theta_i x_i = \theta^T x = \theta x T$$



Multiple Linear Regression

- Cost function:

$$J(\theta_0, \theta_1, \ldots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Simple Linear Regression, n=1

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial \theta_0} J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update θ_0, θ_1) $\}$

Multiple Linear Regression, n>1

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

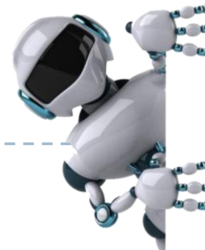
$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$\ldots$



OLS for multiple features

- Cost function with matrix notations:

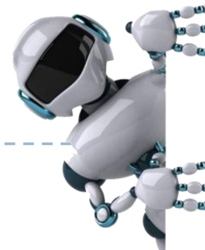
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 = \frac{1}{2m}(X\theta - y)^T(X\theta - y)$$

- Where

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \qquad y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

- Derivatives with respect to θ $\nabla_\theta J(\theta) = X^T X\theta - X^T y$

- By setting them to zero $\theta = (X^T X)^{-1} X^T y$



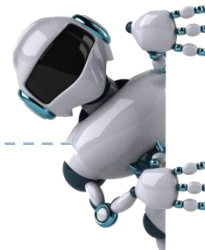
When to use OLS or GD ?

- The following is a comparison of GD and the OLS (normal equation):

Gradient Descent	OLS (Normal Equation)
Need to choose alpha	No need to choose alpha
Needs feature scaling	No need for feature scaling
Needs many iterations	No need to iterate
$O(kn^2)$	$O(n^3)$, need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

- $n=10^4-10^5$ is usually the threshold of choosing GD over OLS.
- Feature scaling helps converting the features to the same scale.
- For example, if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then,

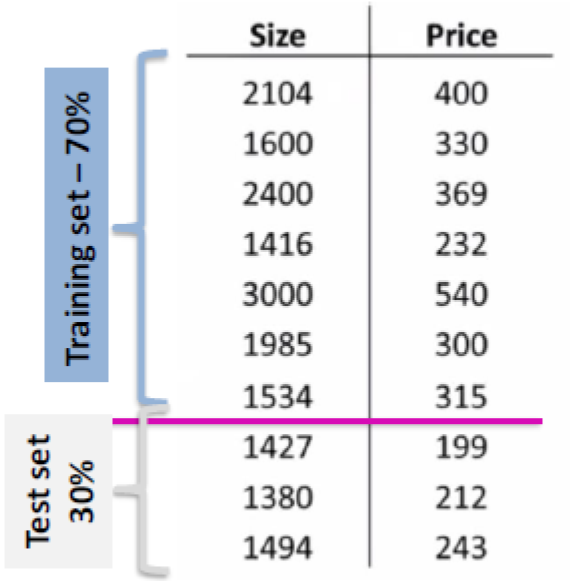
$$x_i = \frac{price - 1000}{2000 - 100}$$



1.4
Assessing Performance

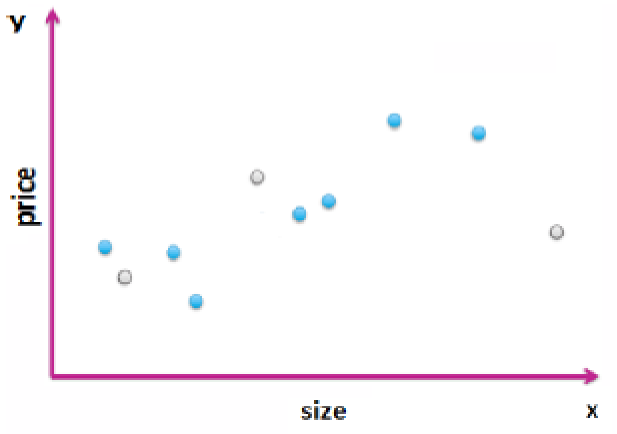
Assessing Performance

- This is about knowing how well the model will **generalize** to unseen data.
- One of the most used methods is splitting the data into train/test sets.

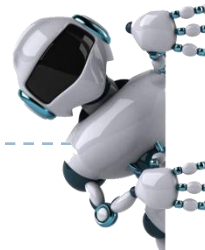


Training set – 70%
Test set 30%

Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

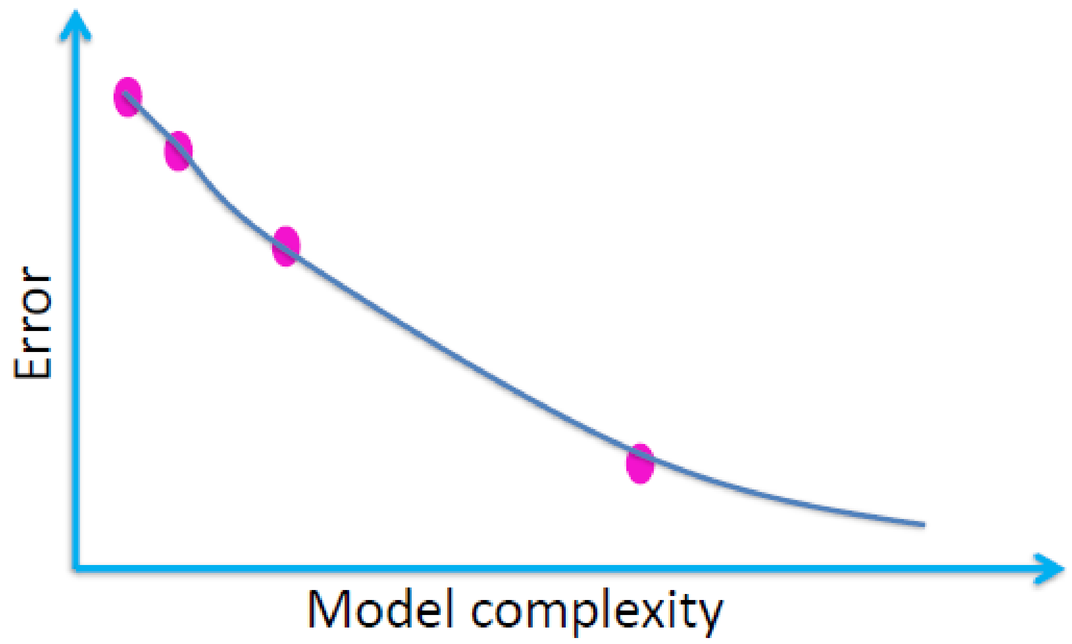


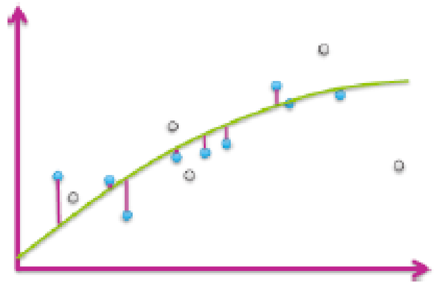
- Learn θ from training data (minimizing training error $J_{train}(\theta)$).
- Compute test error $J_{test}(\theta) = \frac{1}{2mtest} \sum_i^{m_{test}} (h_\theta(x^{(i)}) - y^{(i)})^2$

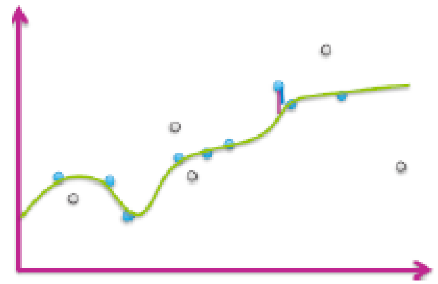


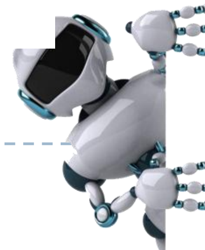
Training error vs. model complexity

- Training error decreases with increasing model complexity



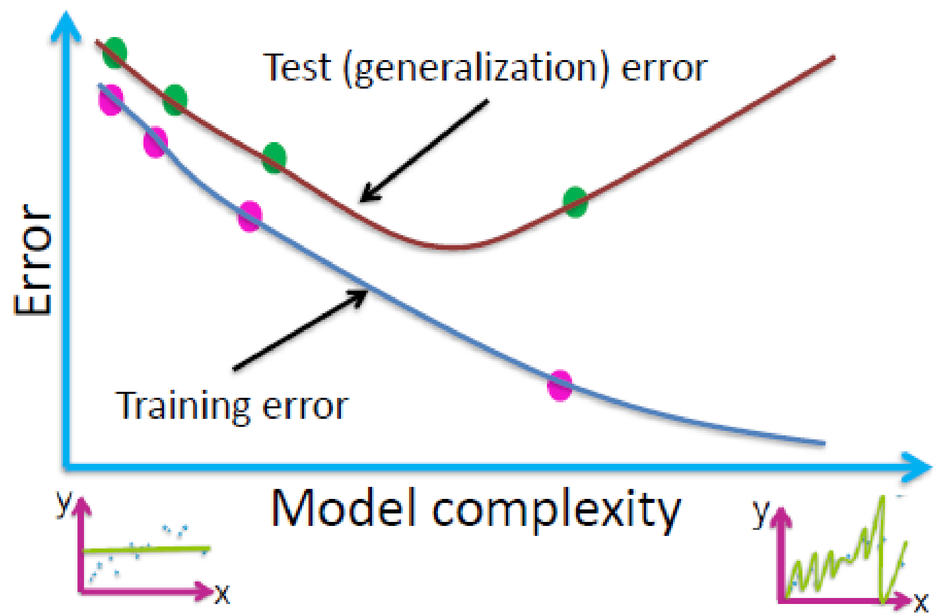


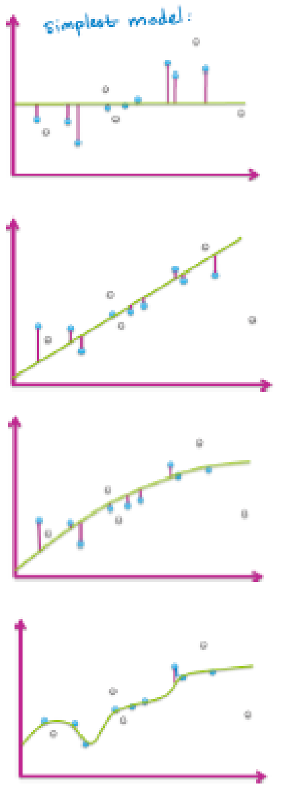


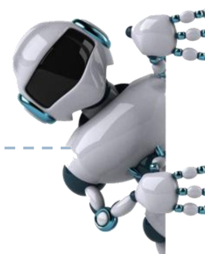


Training error vs. model complexity

- Test error can be used as an approximation of the generalization error

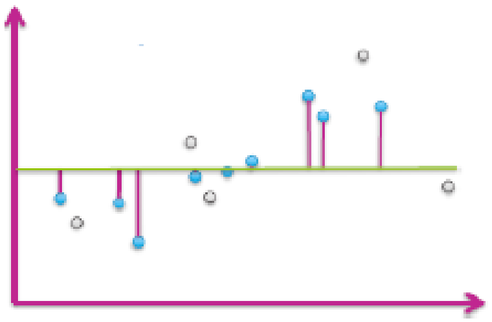


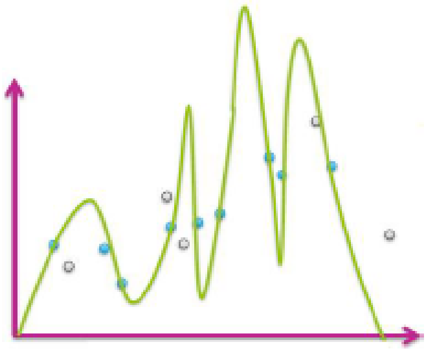




Bias – Variance tradeoff

Model Complexity



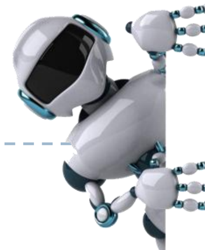


- **High bias model**

The model does not capture enough
The structure of the training set
Parameters tend to be small

- **High variance model**

The model is too specific to the structure
Of the training set
Parameters tend to be very large



1.5
Regularization

Regularization

- It's about finding balance between:
  - How well the model fits the data
  - The magnitude of coefficients
- This is achieved by incorporating a penalty on weights θ in the cost function.

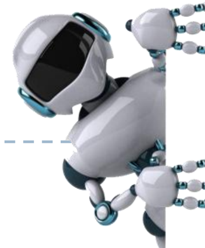
- **Ridge Regression (L2regularization)**

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=0}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} (\theta_j)^2 \right]$$

- **Lasso Regression (L1regularization)**

$$J(\theta) = \frac{1}{2m} \left[\sum_{i=0}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 + \lambda \sum_{j=1}^{n} |\theta_j| \right]$$

- Λ is the regularization parameter:
  - Ridge: Encourages small weights θ but not exactly 0.
  - Lasso: "Shrink" some weights θ exactly to 0.



GD with Regularization

- Reminder of the L2 regularization cost function:

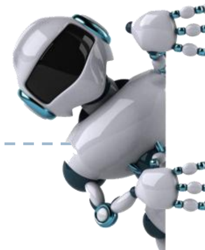
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

- Previously:

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha \quad \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

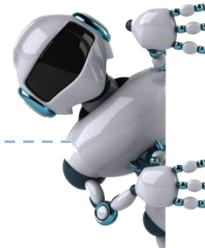
- With regularization: $\theta_j := \theta_j(1 - \alpha\frac{\lambda}{m}) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
- α, λ are learning parameters to choose manually
- In practice: $(1 - \alpha\lambda/m)$ is between 0.99 and 0.95



Other Linear Regression Models

- Number of visiting customers to a website.
- Product demand, inventory, failure, …
- Stock pricing.
- Insurance claims severity.

"Remember that all models are wrong;
the practical question is how wrong do they
have to be to not be useful."
George Box, 1987



Thank you for your attention

Practical work

LAB2: Back in 15min!