Regression: Linear Regression

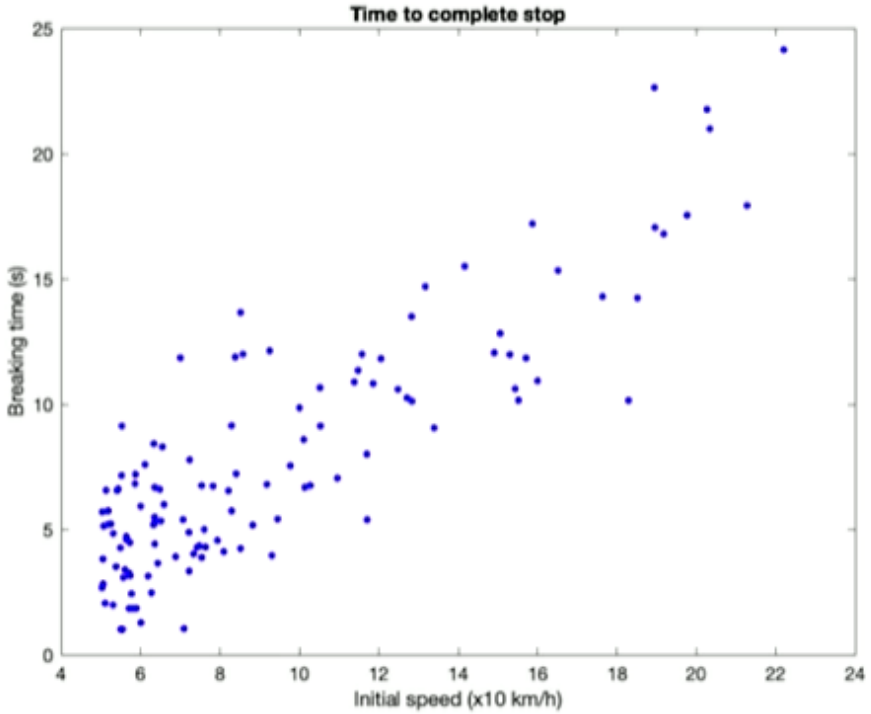
Supervised Learning: When You Know the Answer

You know the right answer, but there is too much data for you to produce the right answer for each case input into the system

- Example: How long does it take for a car to break to a halt?
- Depends on car weight, brake types / states, tires, road, humidity…
- How do you do that?

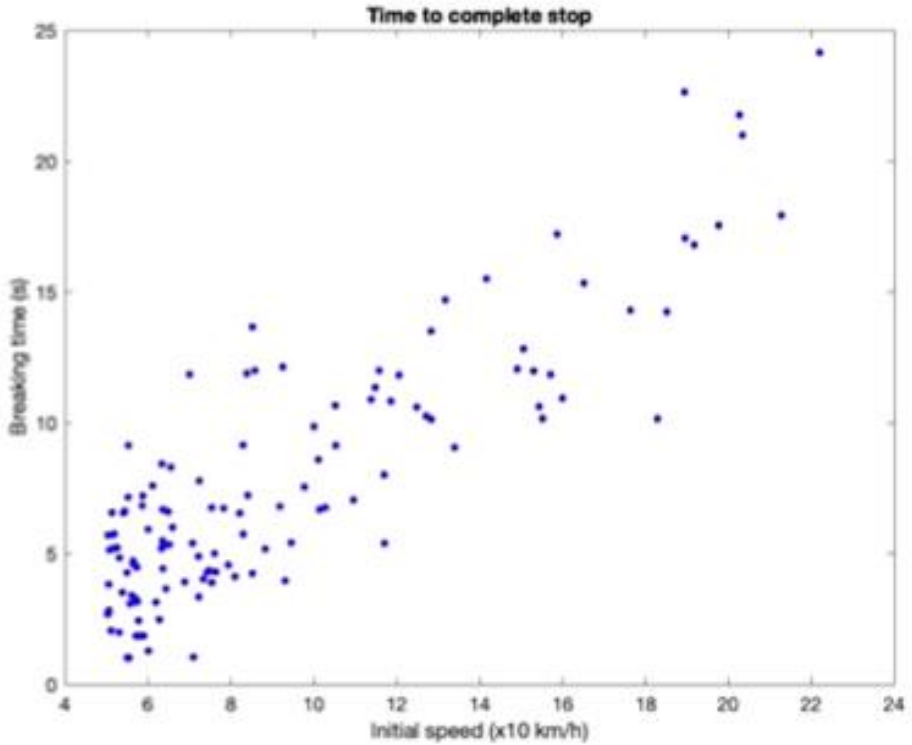
Linear Regression

1. You collect data about a car stopping from different speeds, and you plot these points – to make the process more visual, let's consider the time to stop required, depending on the initial speed.
2. Then you let the machine find the relationship speed and time needed.
3. Next, if you know the speed, you should know how long it will take to stop.



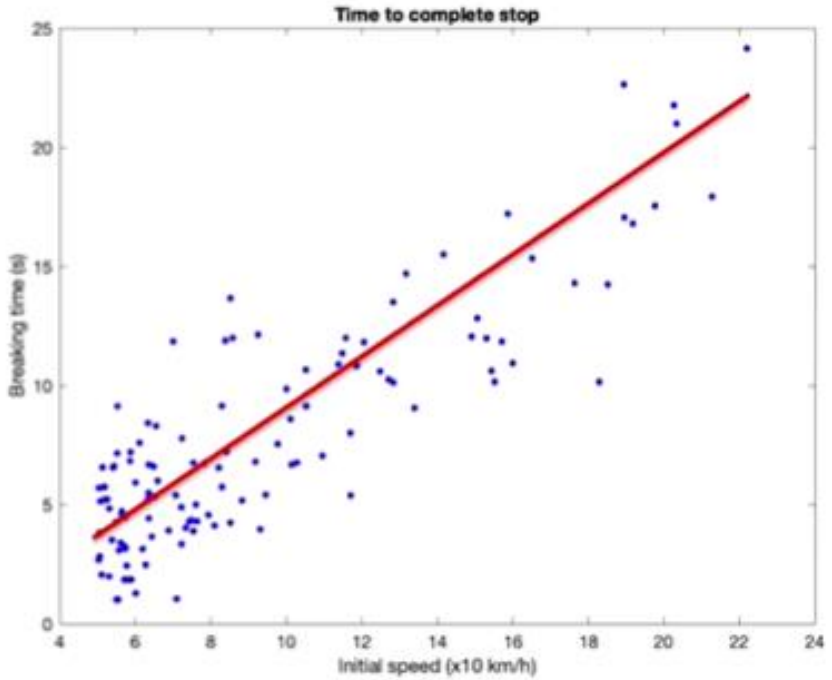
Why Linear Regression

- Visually, it is clear that there is a direct relationship between the speed and the time needed to stop



Why Linear Regression

- Visually, it is clear that there is a direct relationship between the speed and the time needed to stop
- The relationship is "linear"*
- You can draw a line, but could a computer draw that line?



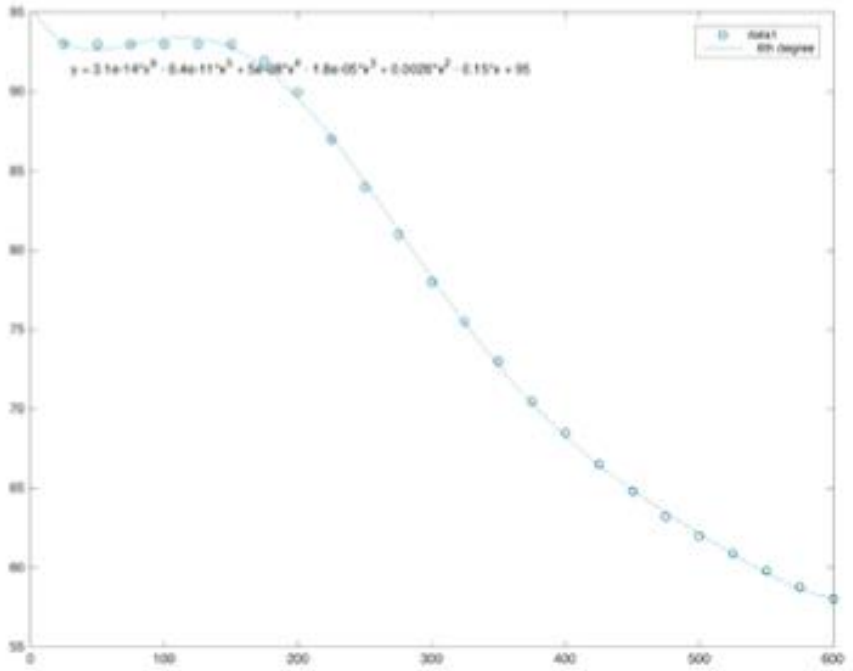
* That line is not "perfect" for all points, but it is the "closest" to all points

Supervised Learning Can Get Complex

Supervised learning can get complex -> non-linear and in many dimensions

$$y = 3.1^{-14}x^6 - 6.4^{-11}x^5 + 5^{-8}x^4 - 1.8^{-5}x^3 + 2.6^{-3}x^2 - 0.15x + 95$$

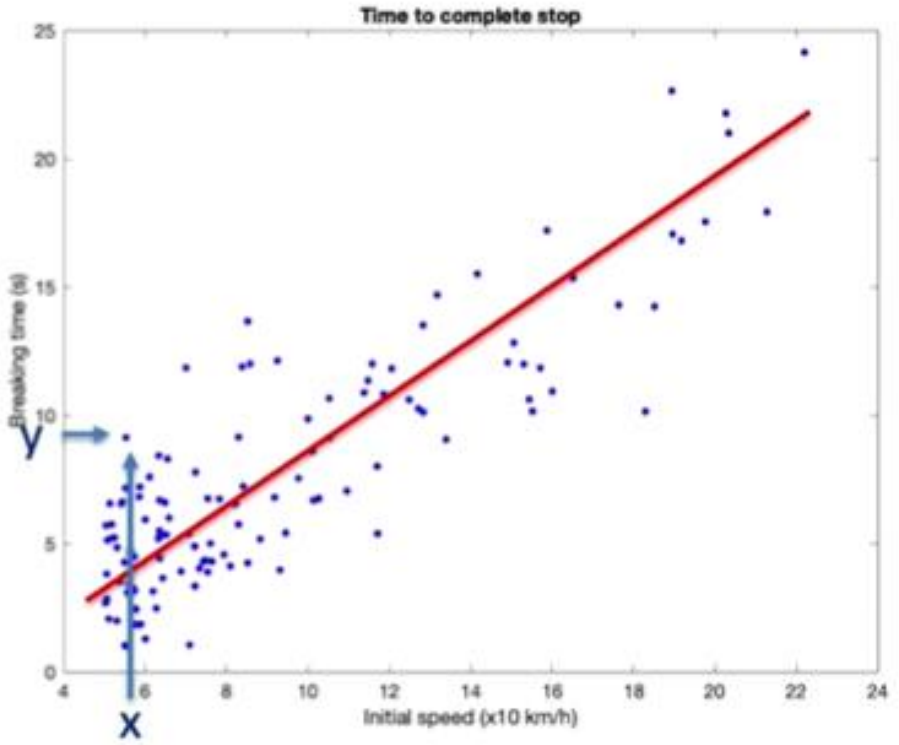
- The equation of the line you try to find can be complicated
  - It is still regression, but not "linear" anymore
- The variables you use, their exponents, etc. are called "features"
- Feature engineering is the primary role of the data scientist



How Linear Regression Works

Each blue point has coordinates (x,y) that you know from your dataset;

x → 6.1101,7.592
5.5277,9.1302 ← y
8.5186,13.662
7.0032,11.854
5.8598,6.8233
8.3829,11.886
7.4764,4.3483
8.5781,12
6.4862,6.5987
5.0546,3.8166



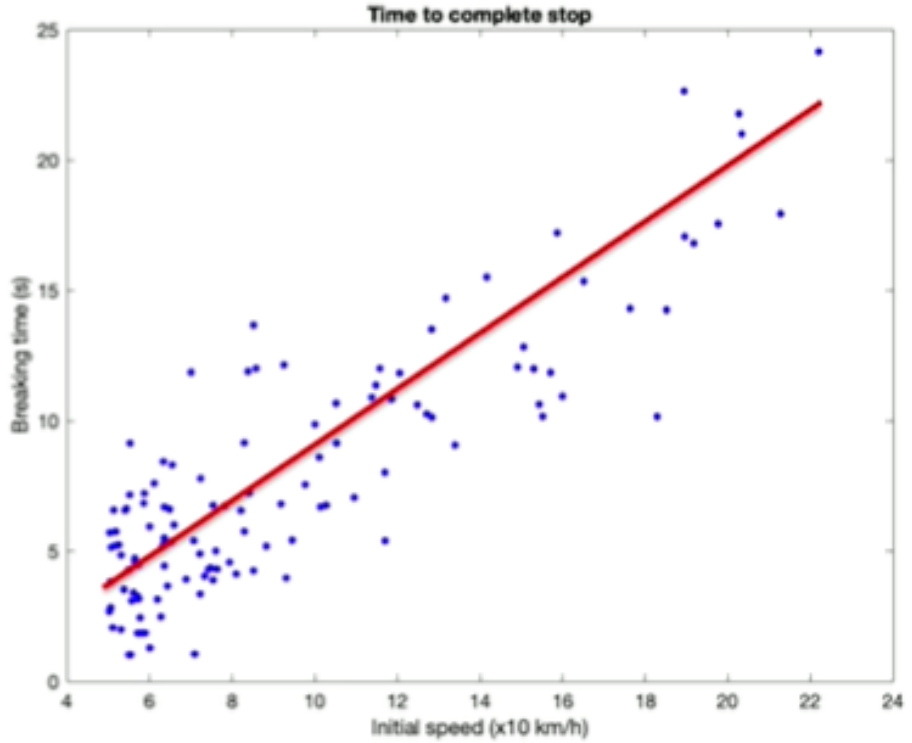
Describing a Line

Let's do some math.

The red line is an equation

y = ax + b

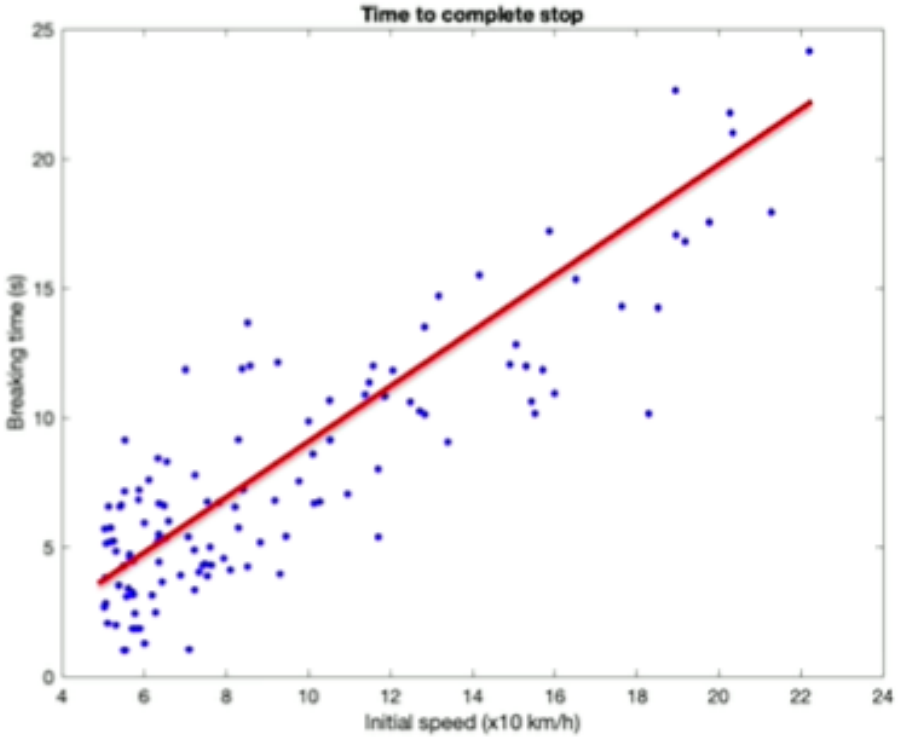
(in ML, we say $\theta_0 + \theta_1 x$)



Line Equation in Machine Learning

$$y = \theta_0 + \theta_1 x$$

Means that if you take an x value



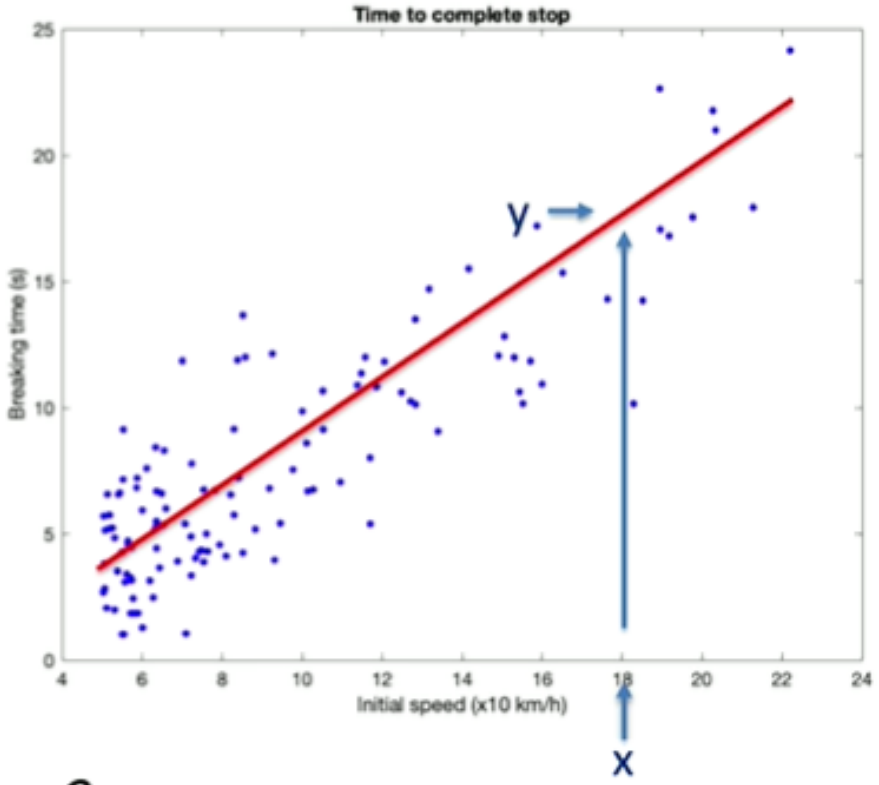
Line Equation in Machine Learning

$$\mathbf{y} = \theta_0 + \theta_1 x$$

Means that if you take an x value

Then compute $\theta_0 + \theta_1 x$

You should find the y on the line



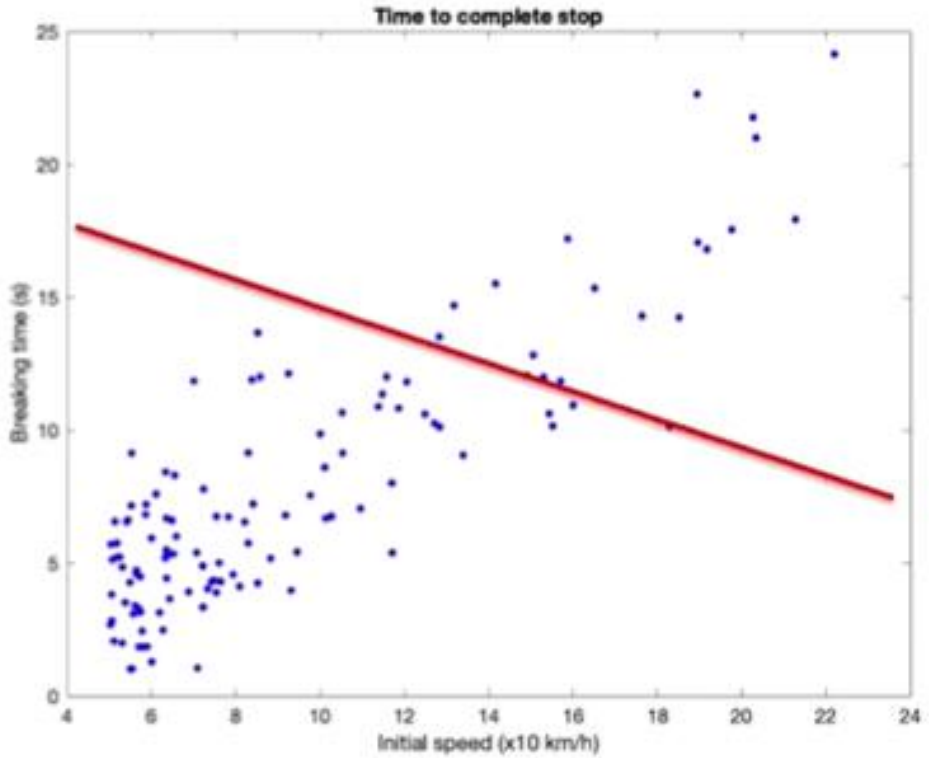
The issue is that you do not know θ_0 or θ_1

Finding the Equation – Random Start

Take two random θ_0 and θ_1

(in most cases, 0 and 0)

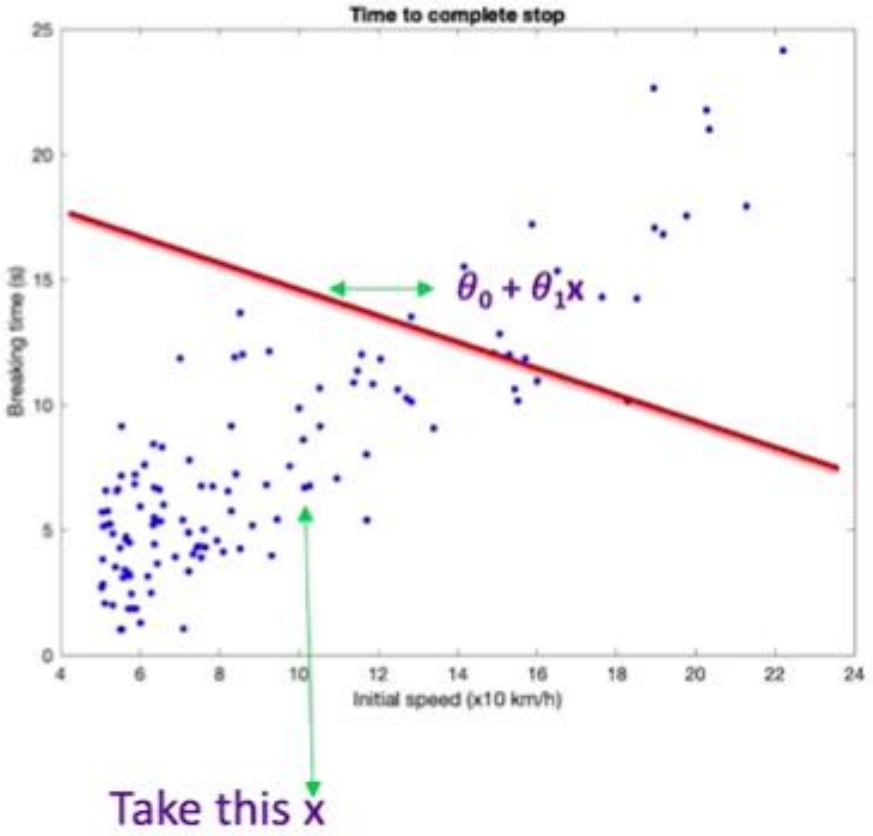
Your line is likely to be very wrong at first (but that's okay)



The Hypothesis Function

Then, for every blue dot you have (x_n, y_n)

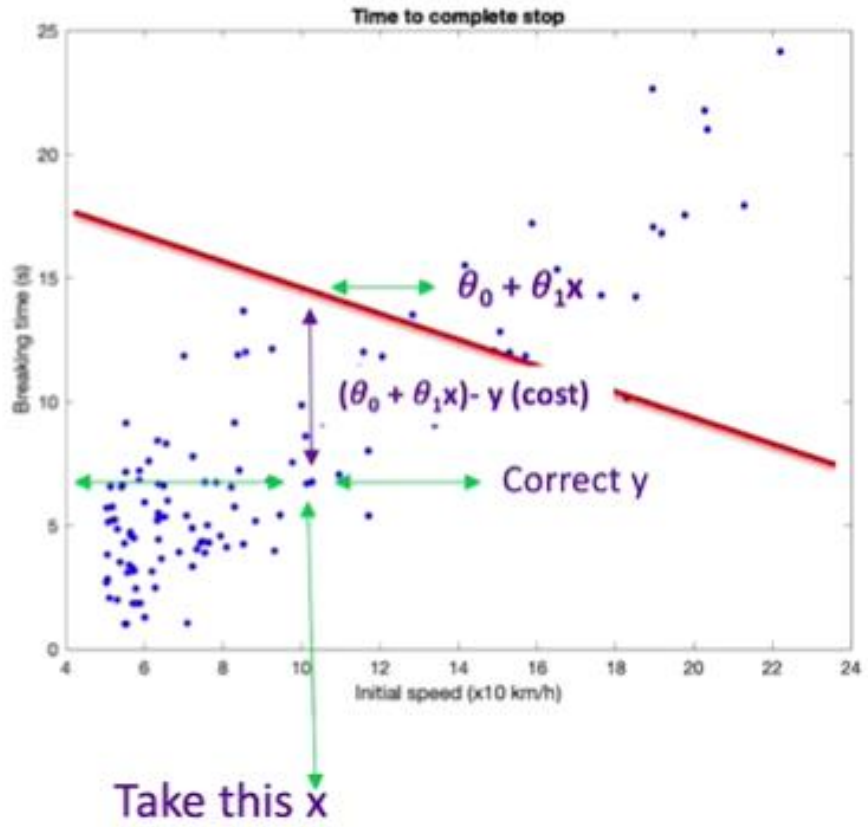
Do $\theta_0 + \theta_1 x$ (that's your hypothesis function) and check how far you are from y_n



The Hypothesis Function

Then, for every blue dot you have (x_n, y_n)

Do $\theta_0 + \theta_1 x$ (that's your hypothesis function) and check how far you are from y_n



Compute the Cost of your Line

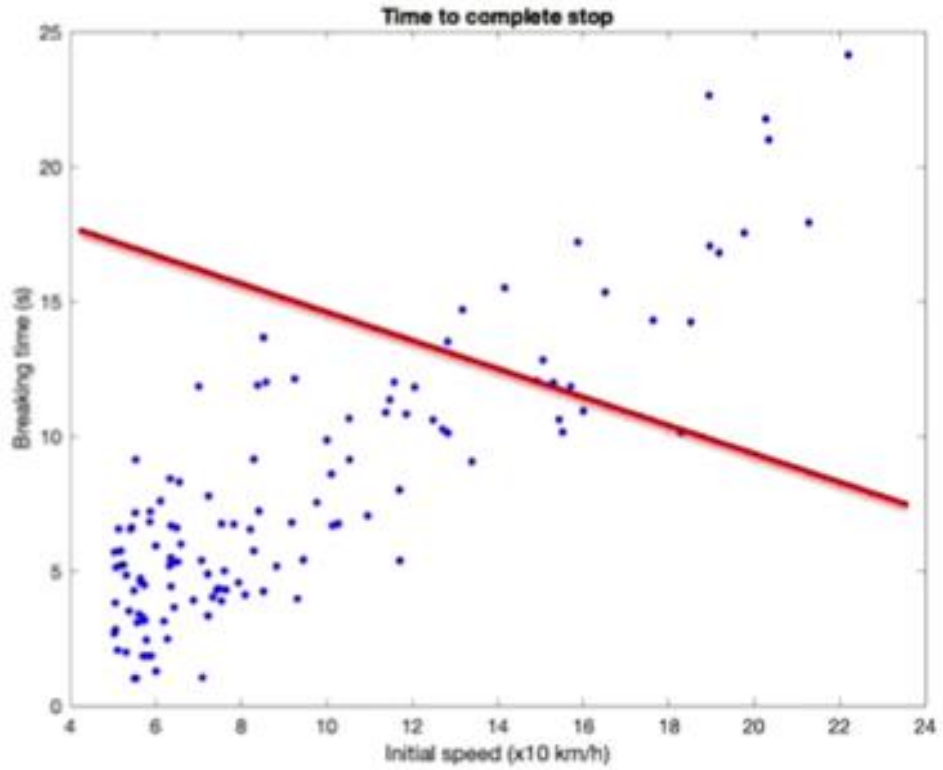
Repeat for each point for which you have (x_n, y_n)

Add all these mini-differences

$$(\theta_0 + \theta_1 x) - y$$

The total is "how far is your theoretical line from the best line for these points"

(your "total cost")



Compute the Cost of your Line

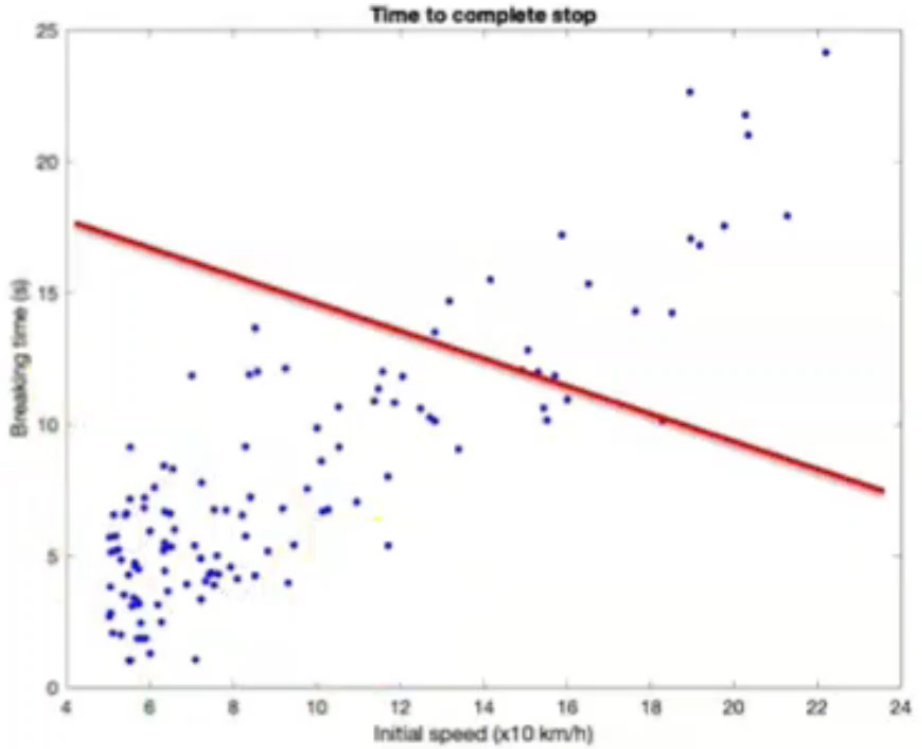
Repeat for each point for which you have (x_n, y_n)

Add all these mini-differences

$$(\theta_0 + \theta_1 x) - y$$

The total is "how far is your theoretical line from the best line for these points"

(your "total cost")



Compute the Cost of your Line

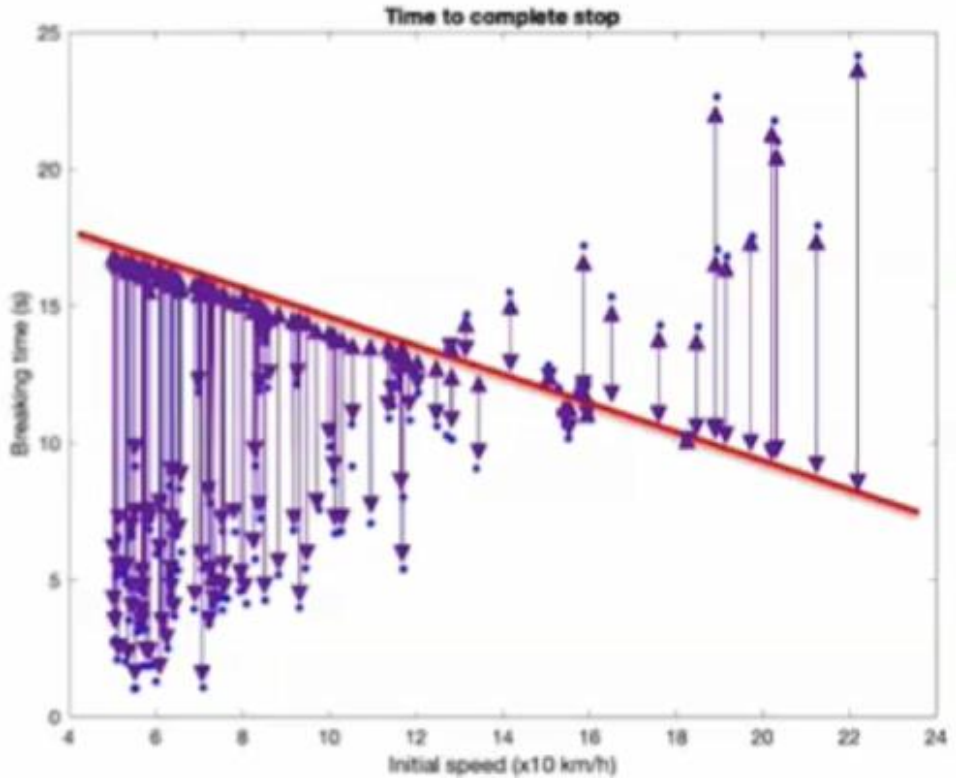
Repeat for each point for which you have (x_n, y_n)

Add all these mini-differences

$$(\theta_0 + \theta_1 x) - y$$

The total is "how far is your theoretical line from the best line for these points"

(your "total cost")



The Cost Function

Another way to say it: the Cost Function is:

$$J(\theta_0, \theta_1) = \frac{1}{2} m \sum_{i=1}^{m} ((\theta_0 + \theta_1 x_i) - y_i)^2$$

One of the **most** important equations in all of ML/AI

Repeat the Process

Then?

Change slightly θ_0 and θ_1

and repeat…

Then ask yourself:

Am I closer to the real y
with the new θ_0 and θ_1 ?

Gradient Descent

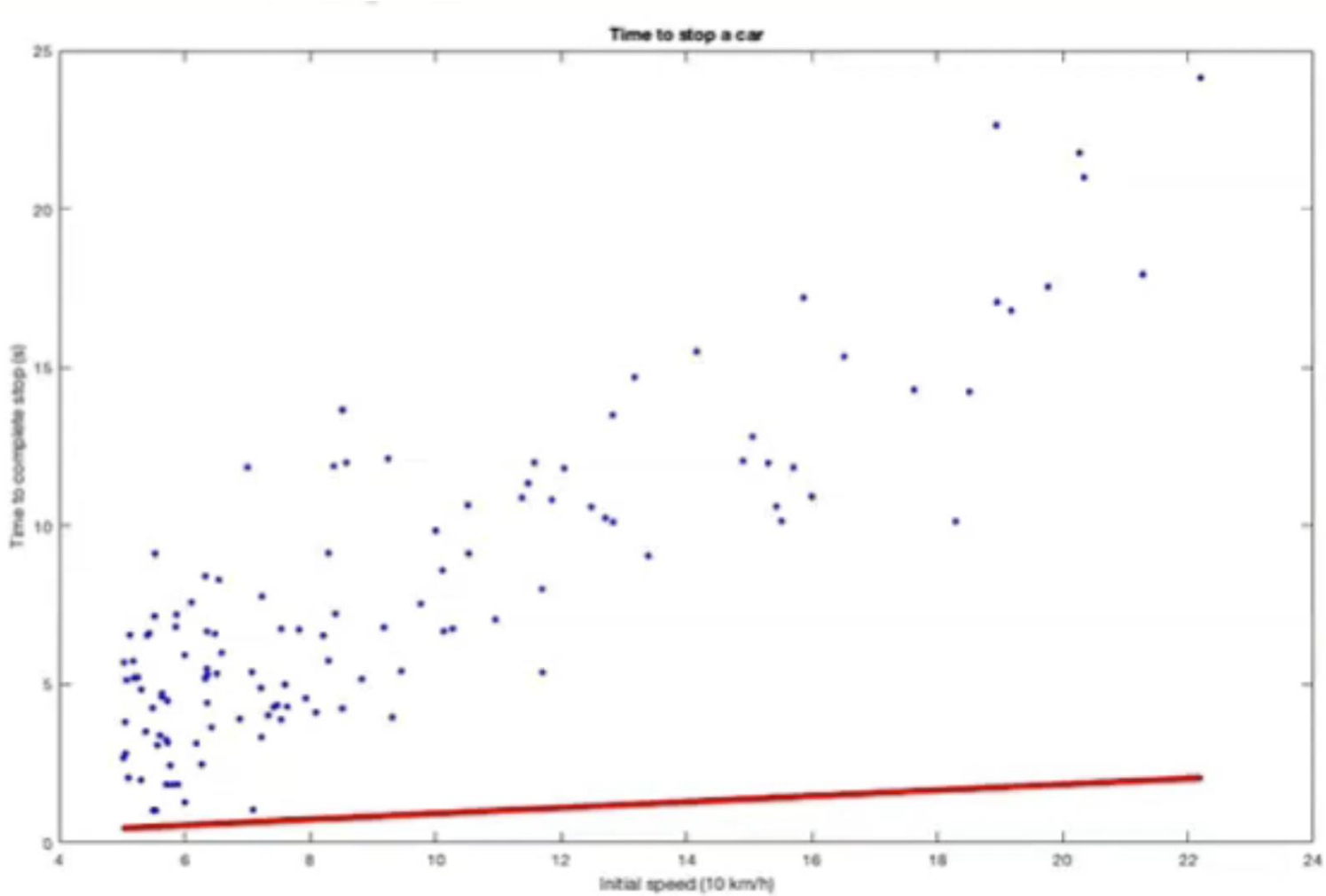
This changing process is called

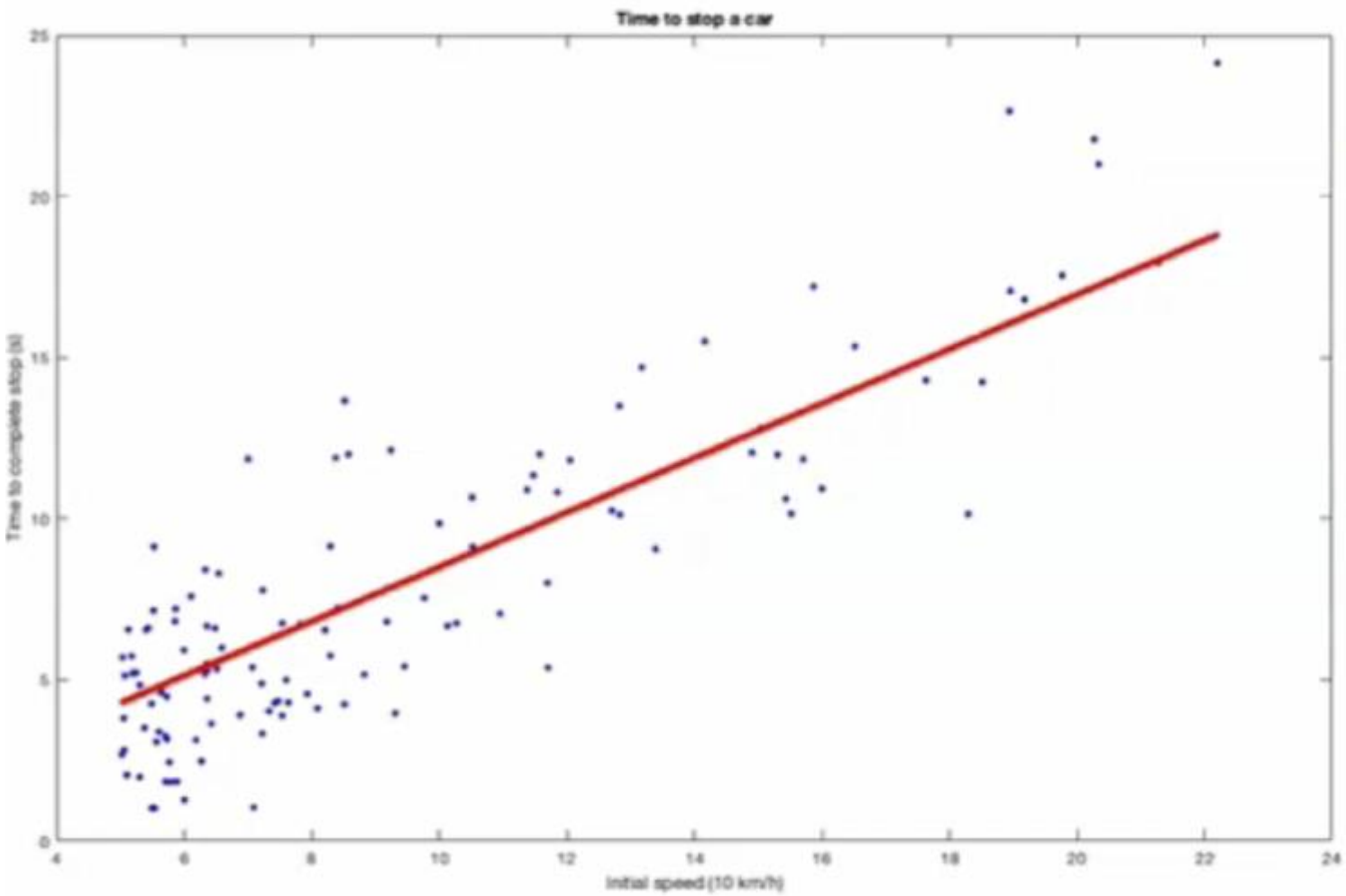
gradient descent

You can call it ***brute-force***

trying all θ_0 and θ_1

until you find the right values


Time to stop a car
Time to complete stop (s)
Initial speed (10 km/h)
0
5
10
15
20
25
4
6
8
10
12
14
16
18
20
22
24


Time to stop a car
Time to complete stop (s)
Initial speed (10 km/h)
0
5
10
15
20
25
4
6
8
10
12
14
16
18
20
22
24

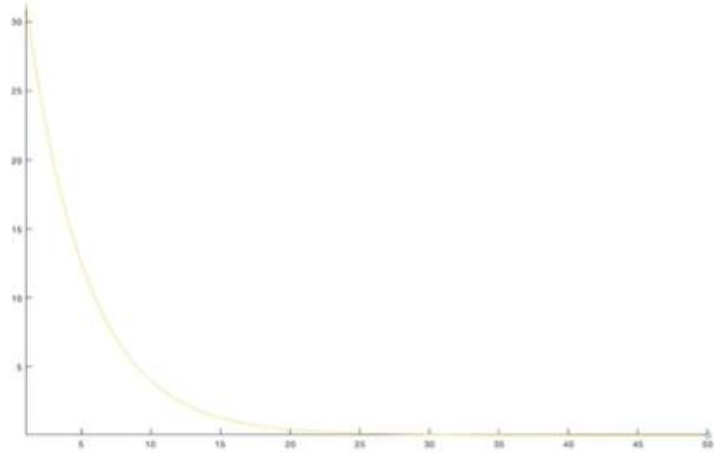
Gradient Descent - A Closer Look

At the beginning, your “cost” is high

Then you get **closer,** and your cost **goes down**

Finding the Right Cost Direction

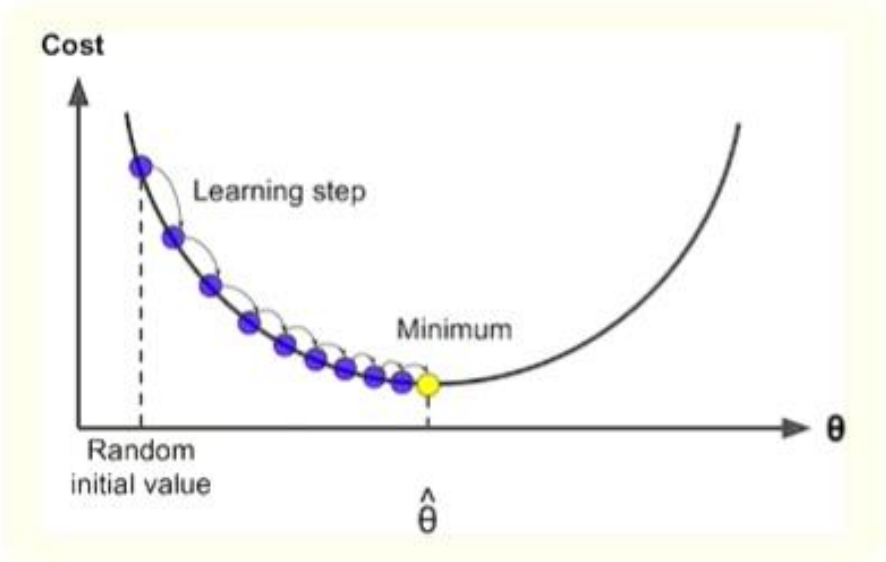
You could graph that cost value



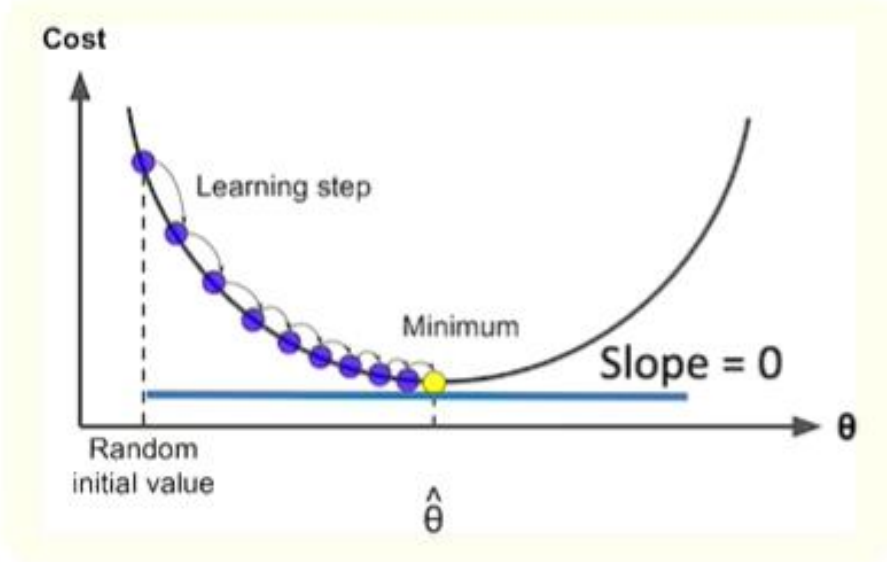
If it’s a curve, it has an equation

Math with "cost" can help find the best next θ_0 and θ_1

We can use Calculus and derivatives



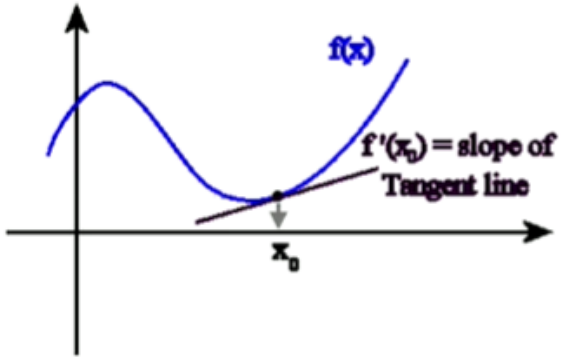
When slope is 0… it means you are at the local minimum of the cost function



When slope is 0… it means you are at the local minimum of the cost function

Use Calculus for Multi Dimensions Graphs

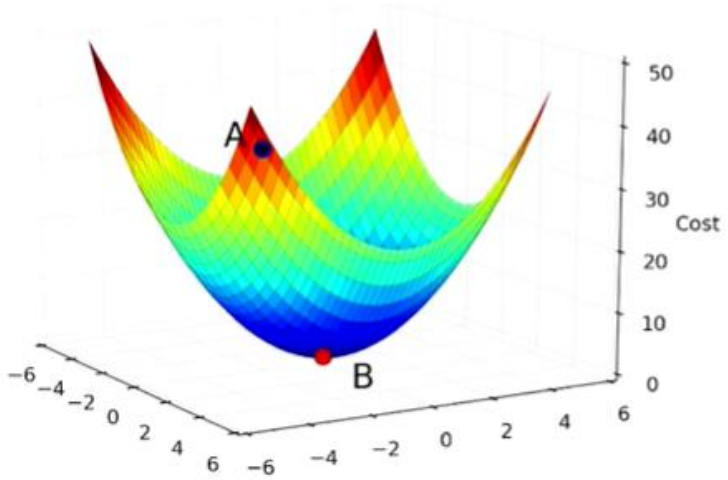
The "cost" is the change in the θ_0 and θ_1 values



Measuring change is what calculus (and derivatives) do!

Gradient Descent In Higher Dimensions

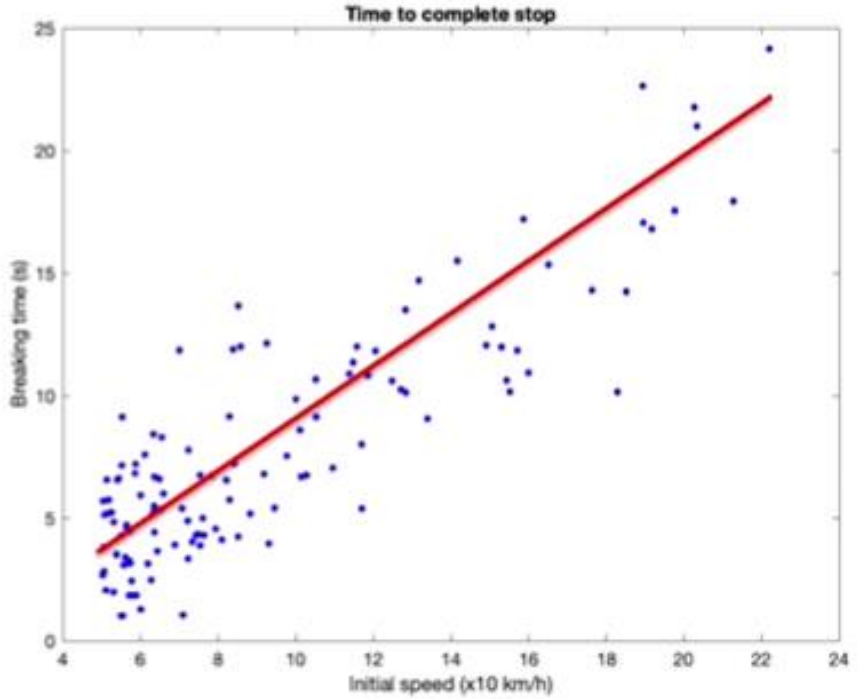
Using derivatives of the line equation gives us the right direction toward the best θ_0 and θ_1



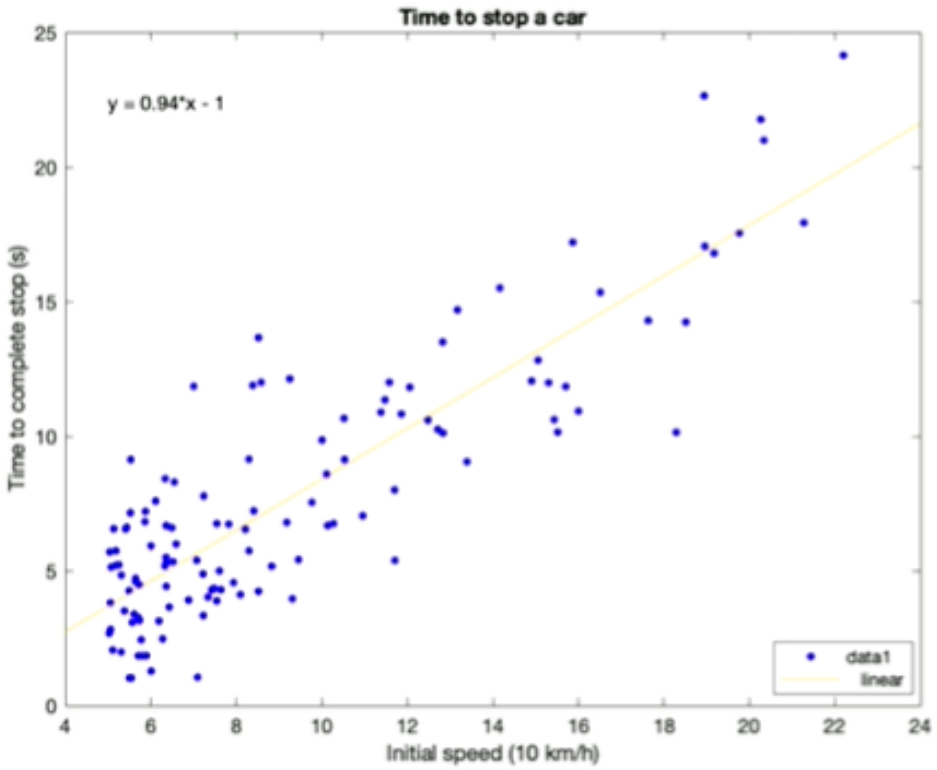
Especially useful in larger dimensions

Linear or Non-linear?

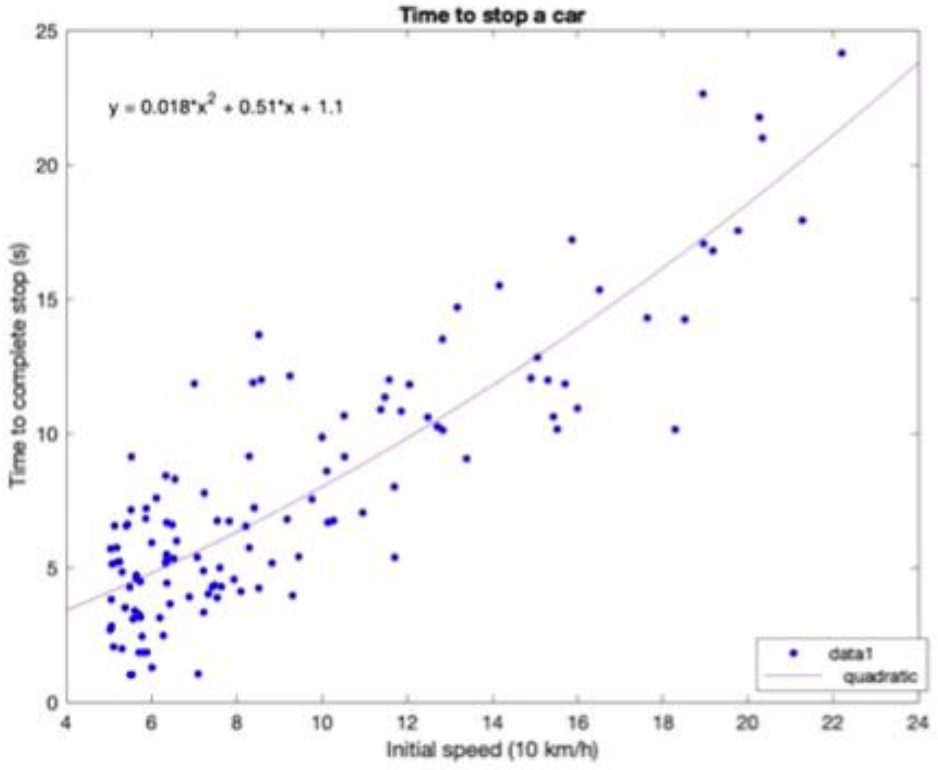
- We tend to love linear regression
  - Simple, easy to visualize & compute
- But we do recognize that it only imperfectly fits our data
  - Looks like a simplified version of our points



Can We Try a Closer Fit?

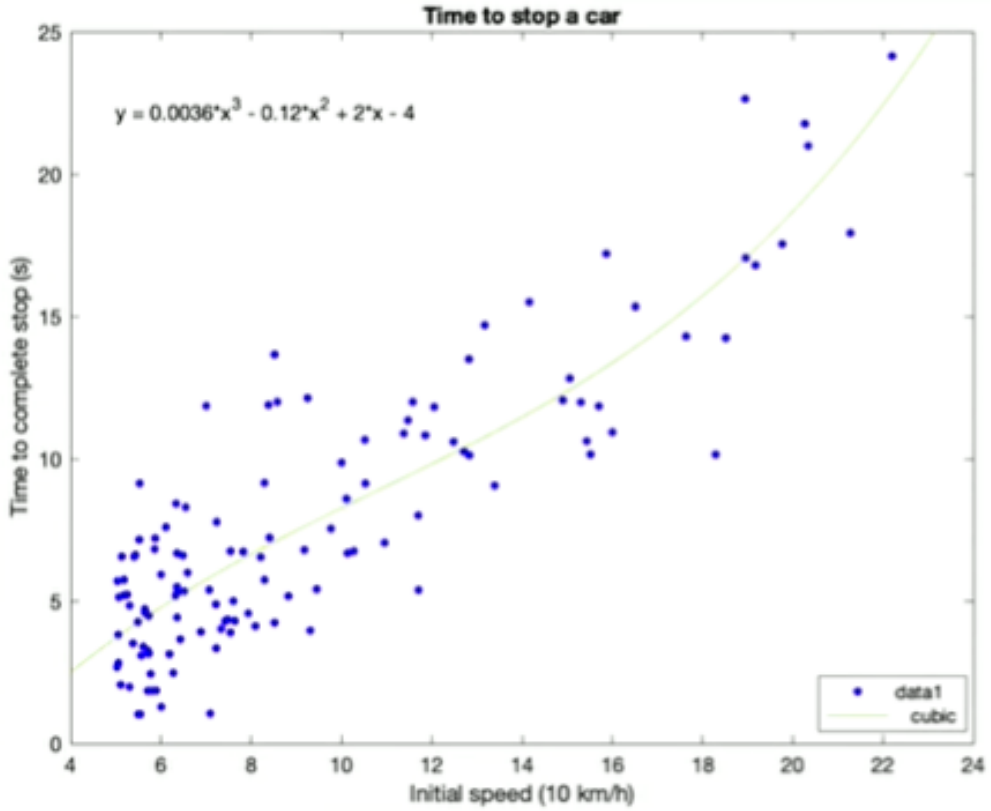


Linear fit

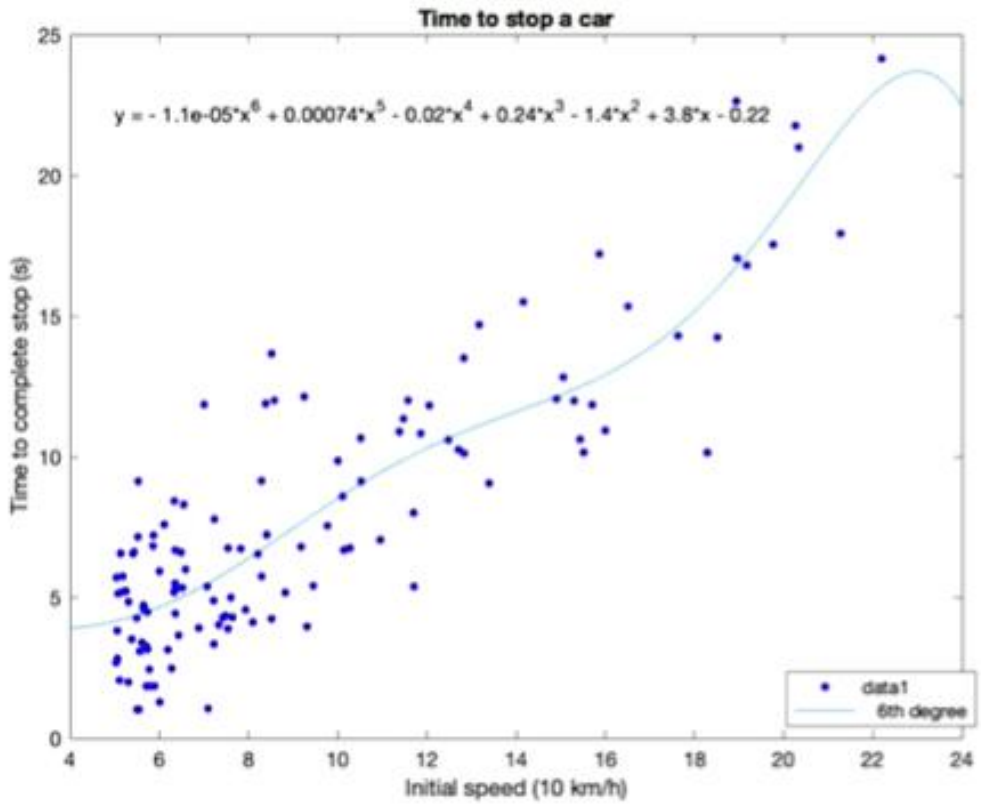


Quadratic fit

Can We Try a Closer Fit?

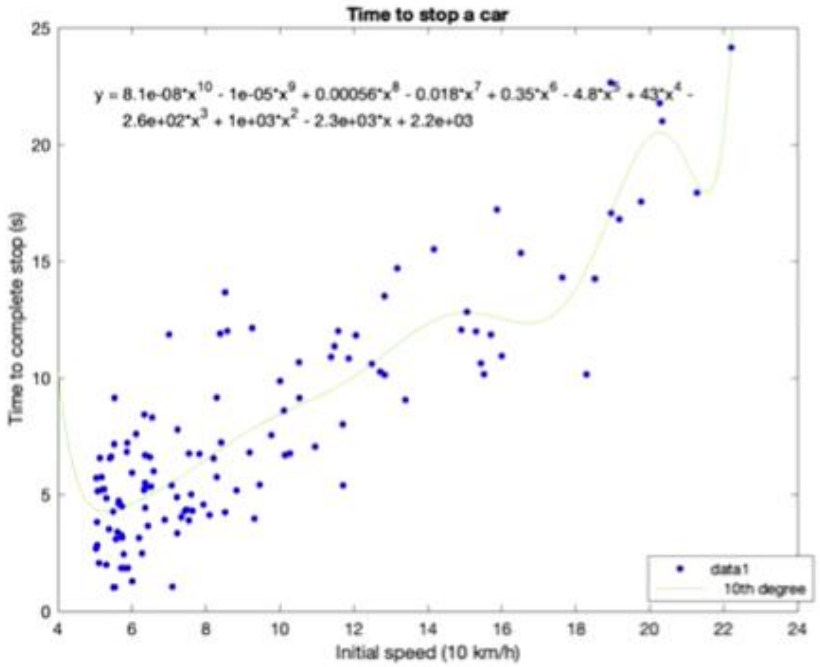


Cubic fit



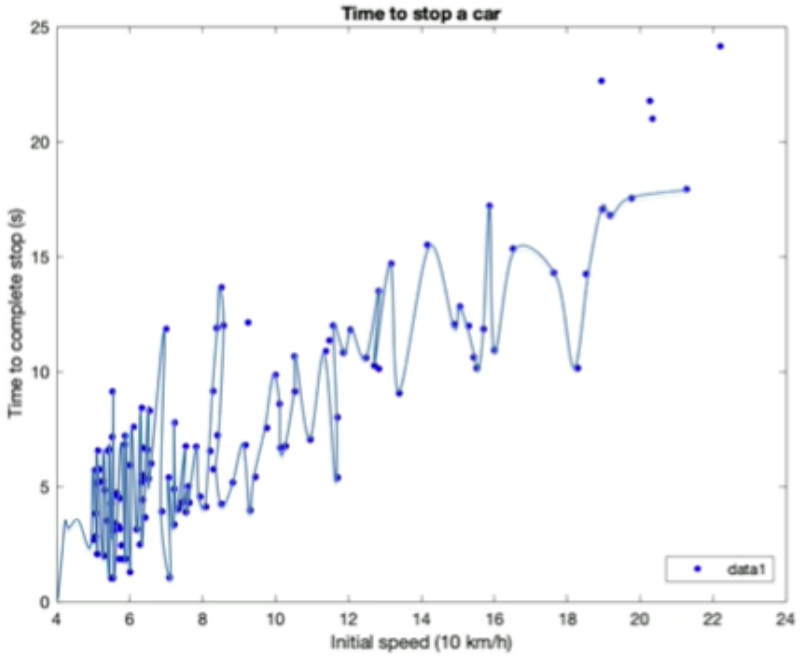
6th degree polynomial fit

Can We Try a Closer Fit?

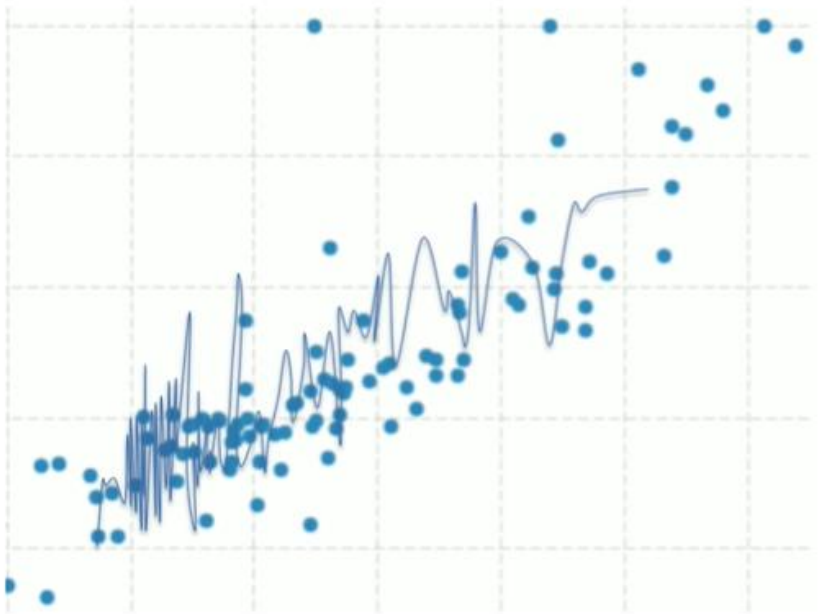


10th degree polynomial fit

There Is No Limit to the Fitness Attempts…



Until You Apply Your Equation to the Real World

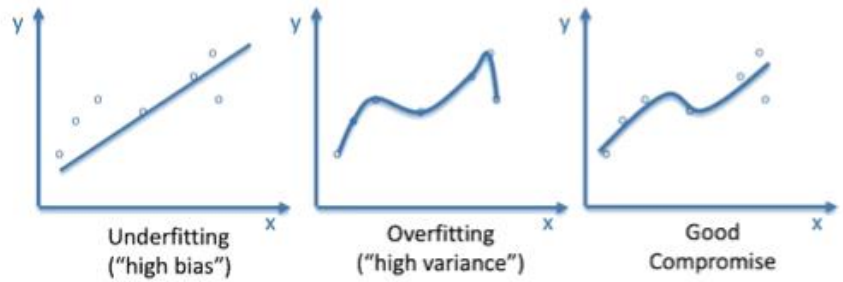


Real World data might be different from your training set

This issue is called ***overfitting***

Finding the Right Line

The main challenge in Supervised Learning is to find the right equation… and figure out if the samples represent the full population



Use 2 or 3 Sets

To avoid the high bias / high variance issues, divide your data set into 3 groups:

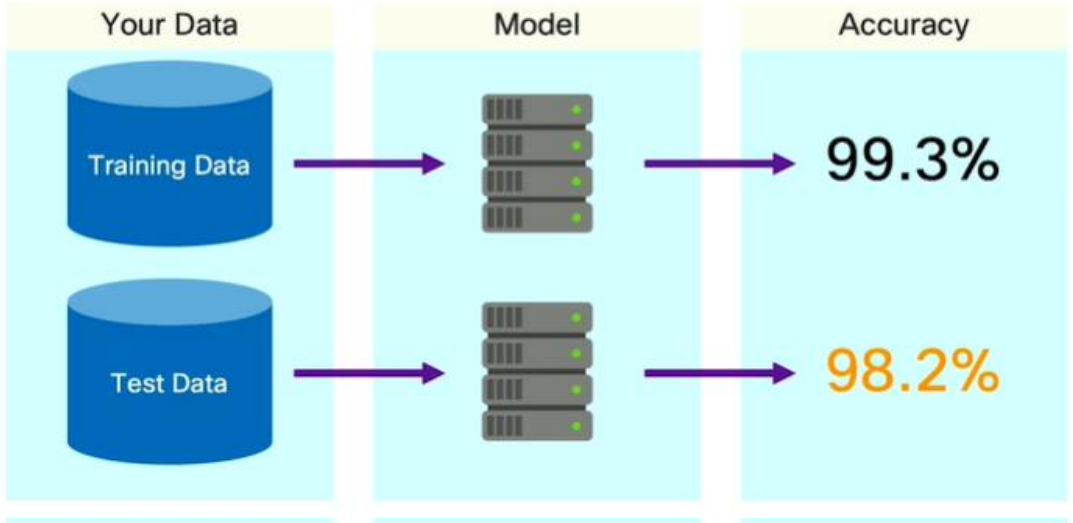
- Training dataset
  - E.g. 60% of your data, train your model
- Validation dataset
  - E.g. 20% of your data, verify and refine your model
- Test dataset
  - E.g. 20% of your data, confirm your model
  - If the model does not work, rework it!

Reduce Your Dimensions

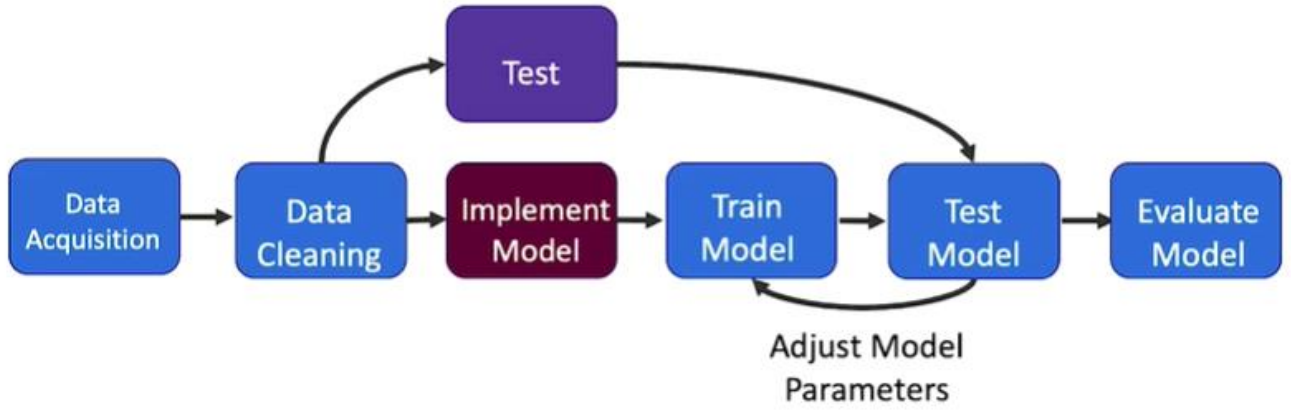
- Reducing the size of the training set speeds up model computation
  - But too small is dangerous too (not representative)!
- Another common technique is to attempt to reduce the number of dimensions
  - E.g. from 5K dimensions to 500
- Principal Component Analysis (PCA) helps do just that

How Accurate is Your Model?

Measure your accuracy!



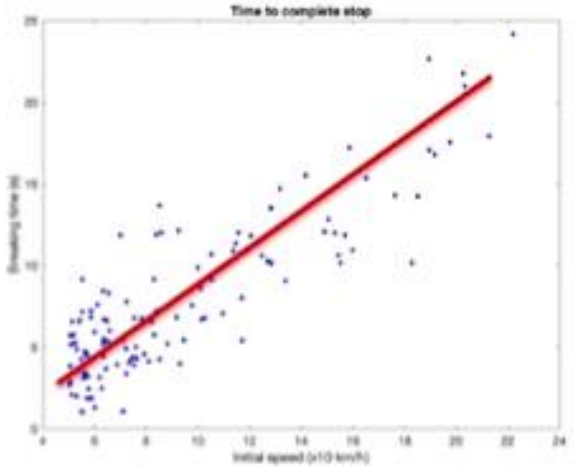
In Summary: A Typical Supervised Learning ML Workflow



Classification: Logistic Regression

Logistic vs. Linear Regressions

- we examined linear regression
  - The line can be straight or curved
  - The goal is to find the line that best fits the data
- Data has some characteristics we try to find and graph



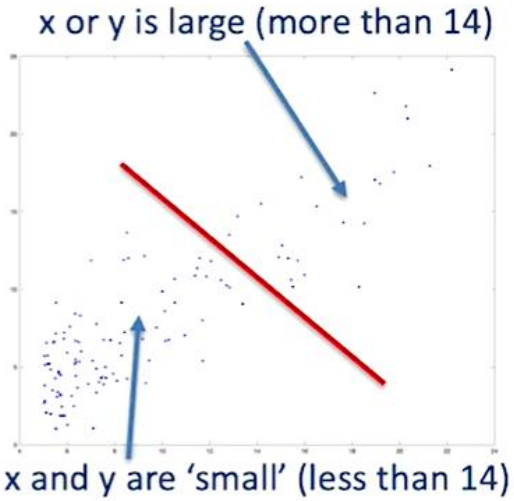
Logistic vs. Linear Regressions

- In other cases, data represents different categories
- Your goal is now to split the data points in groups
  - E.g. finding clumps in a pipe that are both large and hard
  - Organizing data in groups is called classification

ML gotcha:

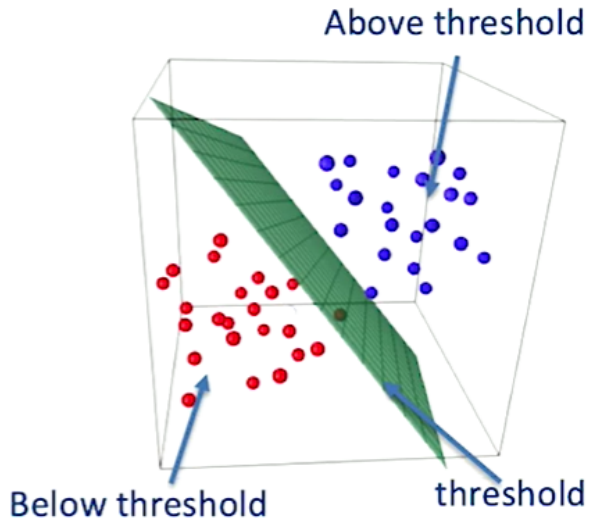
Algorithms that perform classification are called 'logistic regression' algorithms

Different from data description algorithms ('linear regression')



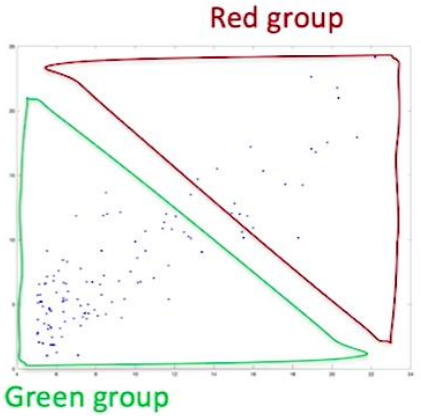
Principles of Logistic Regression

- Logistic regression (simple models) uses thresholds:
  - Apply an equation to your data, choose it cleverly
    - Choose the right parameters and weights
    - You can get a line of separation
  - Data Points which output result below threshold are in one category, above threshold in another category

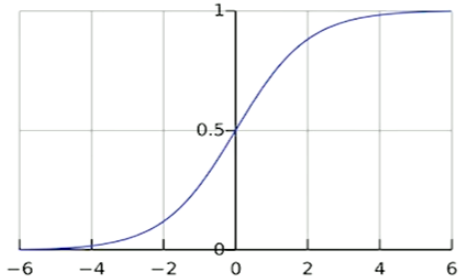


A Probabilistic approach

- With classification, you are not trying to compute the right ‘y’ for a given ‘x’
- You just want the answer “green group or red group”?
  - Sometimes with a likelihood indication
  - This binary logic is great, because you can decide if ‘1’ means ‘red’ or ‘green’

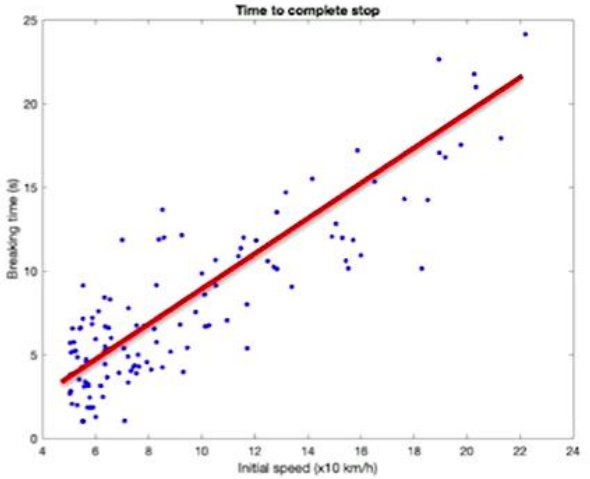


- Group membership is a probability
  - We can use an equation that leverages this property:
    - equation output is close to 0 -> group A
    - equation output is close to 1 -> group B
  - This logic also helps with our general reasoning
    - “With these parameters, can I predict that the next input will be more likely in group A or B?”



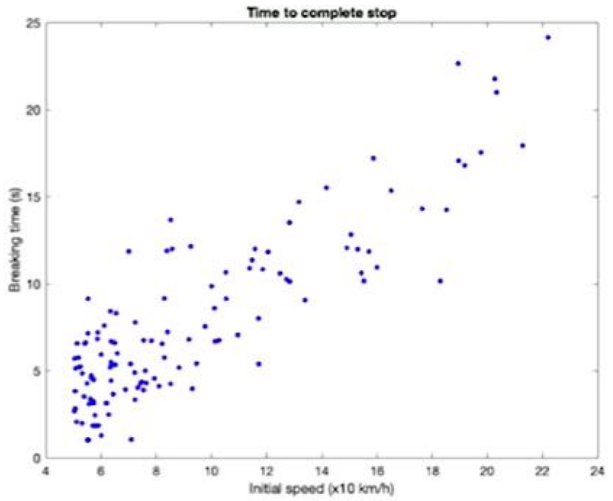
Car Breaking Issue

- In our car breaking example, classification is not “how long before stop based on car speed” but “is the car going to hit the tree?” (yes/no)



Car Breaking Issue

- With linear regression, we found the equation of the line
  - ($y = \theta_0 + \theta_1 x$) (y = -1 + 0.94x)
  - We could design our groups as:
  - Less than 140 km/h -> less than 12.16 seconds to break -> no hitting
  - More than 140 km/h -> more than 12.16 seconds to break -> hitting*

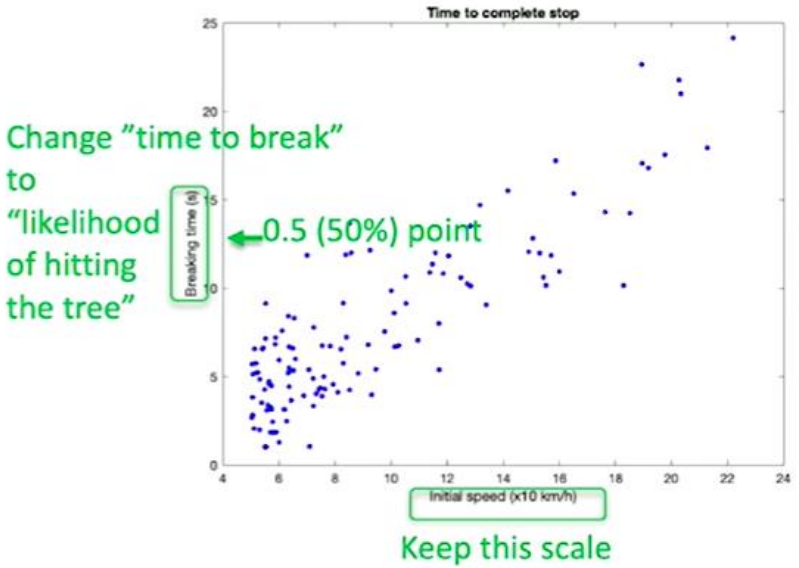


- As this is a probability, we could even say:
- As speed increases, likelihood of hitting the tree increases, and is more than 0.5 (50%) over 140 km/h

* Note that as speed is in tenth of km/h, the time is in tens of seconds, i.e. 120 = 12 seconds

Probability and car break

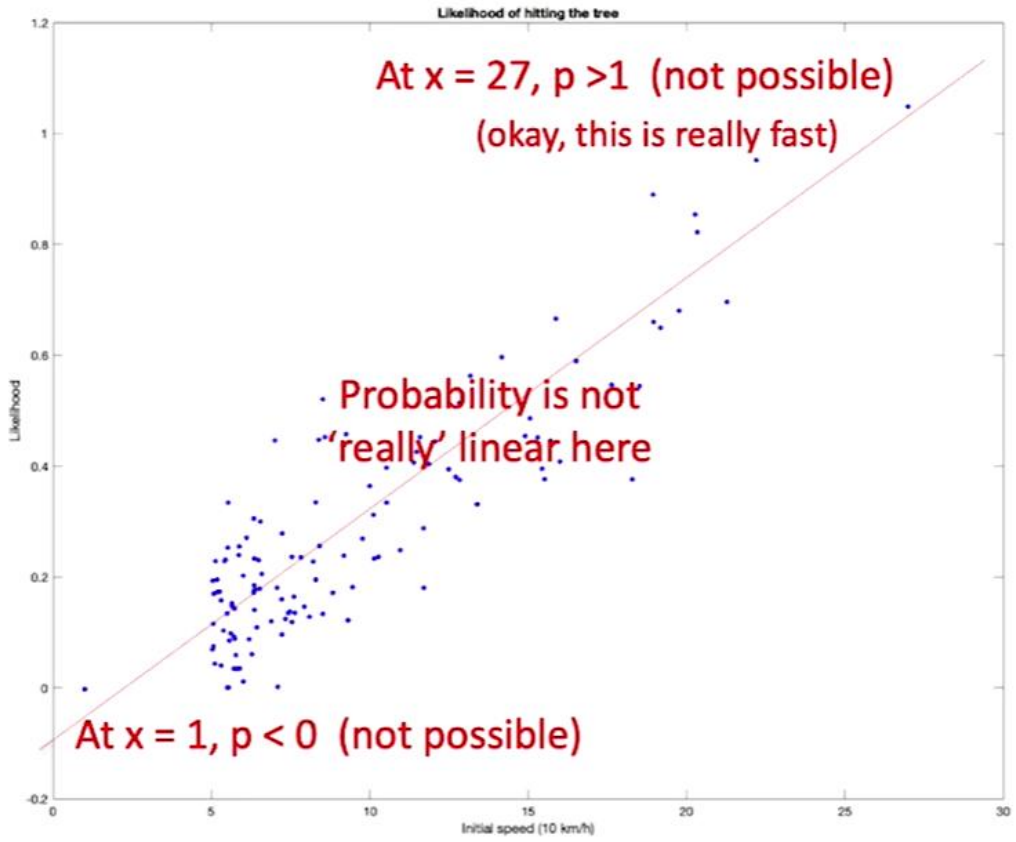
- We could trade our graph scale accordingly
- If we want “12.16” to be our midpoint (0.5, or 50%), then we can just modify our equation:
- $P = \frac{-1+0.94x}{24.32}$ ← 2x12.16



- At x = 14, p = 0.5 (50%)
- At x = 20, p = 0.73 (73%)
- At x = 10, p = 0.34 (34%)
  - All seems to look good

Limitation of the Linear Model

- This “linear” model breaks at the edge:



A Different Equation

- Logistic regression uses a method where:
  - ✓ Output numbers are positive
  - ✓ min is 0 (but not less)
  - ✓ max is 1 (but not more)
- There are a few ways to solve these conditions, but one that works very well:
  - $e^{something}$ is always positive
    - but can be more than 1
  - $\frac{something}{something+1}$ is always between 0 and |1|

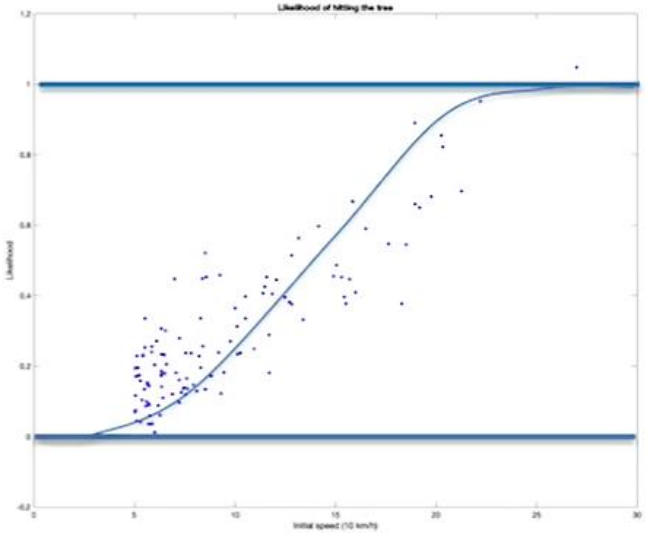
The Logistic Regression Equation

- If we use:

$$p = \frac{e^{\theta_0 + \theta_1 x}}{e^{\theta_0 + \theta_1 x} + 1}$$

- We get the conditions we want!
- It looks complex, but $\theta_0 + \theta_1 x$ is our line equation, and for mathematicians, it is a simple transformation:

$$\ln\left(\frac{p}{1-p}\right) = \theta_0 + \theta_1 x$$



The sigmoid function

- This is too long to write!

$$p = \frac{e^{\theta_0 + \theta_1 x}}{e^{\theta_0 + \theta_1 x} + 1}$$

- Let's do a math trick:
- $e^{\theta_0 + \theta_1 x} = (e^{\theta_0 + \theta_1 x})\ (1)$
- $e^{\theta_0 + \theta_1 x} + 1 = (e^{\theta_0 + \theta_1 x})\ (1 + \frac{1}{e^{\theta_0 + \theta_1 x}})$
- And $\frac{1}{e^{\theta_0 + \theta_1 x}} = e^{-(\theta_0 + \theta_1 x)}$

In general, $\frac{1}{a^n}$ is the same as a^{-n}

Still there?

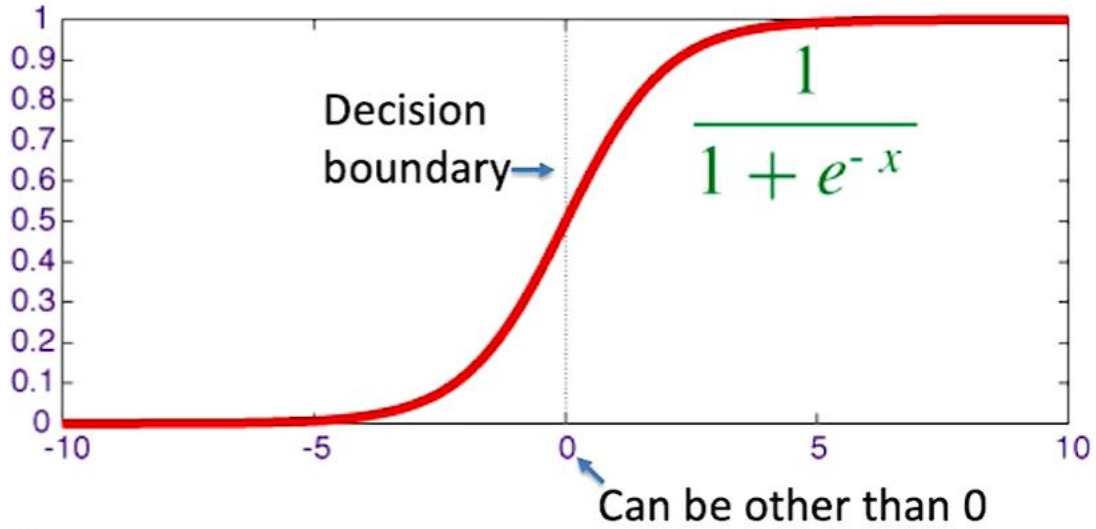
If we take $\frac{e^{\theta_0 + \theta_1 x}}{e^{\theta_0 + \theta_1 x}}$ out, we can write:

$$p = (\frac{e^{\theta_0 + \theta_1 x}}{e^{\theta_0 + \theta_1 x}})\ (\frac{1}{1+e^{-(\theta_0 + \theta_1 x)}})$$

$$p = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

The Sigmoid (or Logistic) Function

It was worth it!



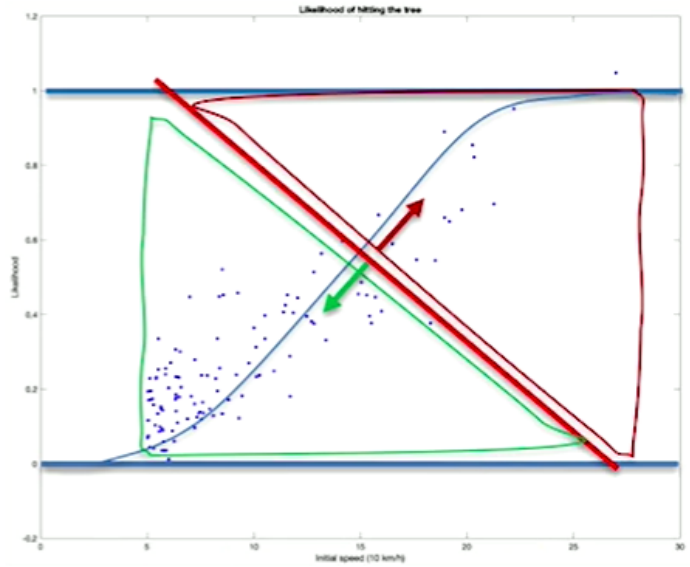
- The sigmoid function is the most common function for logistic regression:
  - Gets your data into 2 groups
  - Tells you the probability of belonging to a group or the other
  - Directly relates to your 'linear' equation

Thresholds and Cost

You can ‘manually’ map “> 0.5 = group A” from our result

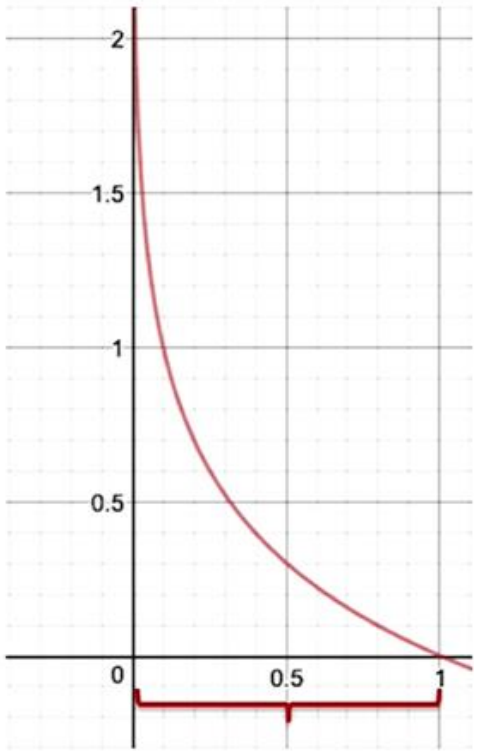
You can also directly use the probability value output

- If you train your model and want a cost, you may want to push points to one side or the other:
  - A point in the green area as a super high red cost
  - A point in the red area has a super high green cost

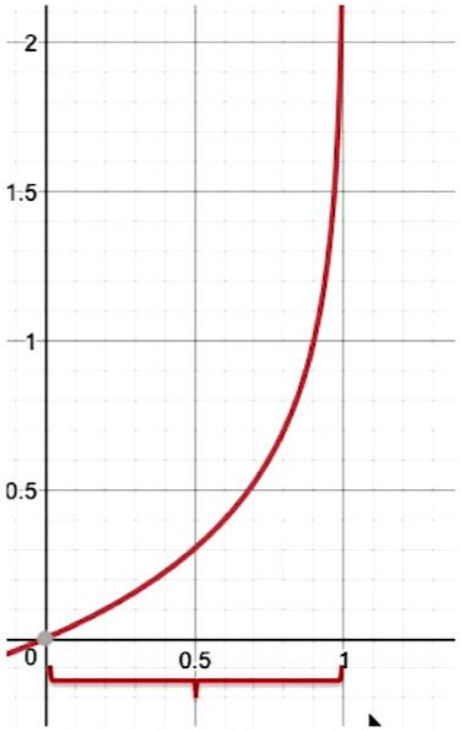


Logistic Regression Cost Function

- A common way to adapt the cost function to the logistic regression work is to use:
  - (remember: $p = \frac{1}{1+e^{-(\theta_0 + \theta_1 x)}}$)
  - -log(p) (cost lower for p=1)



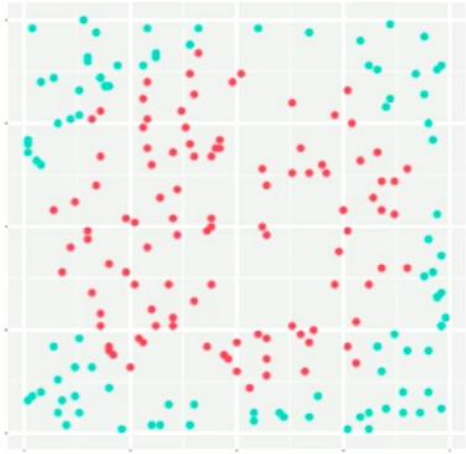
- -log(1-p) (cost lower for p=0)
- Compute both for each p, and find the lowest cost

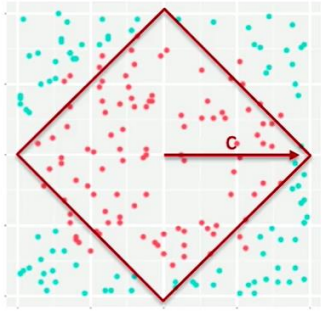


Understanding Support Vector Machines

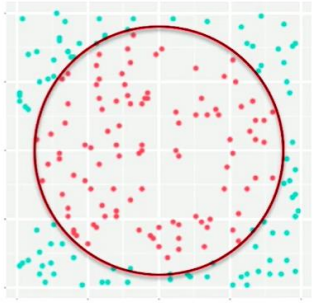
Decision Boundary

- The “threshold” between groups is the decision boundary
  - It can be a simple straight line or a more complex shape
  - Then its equation is going to change accordingly
  - The same ‘best fit challenge’ (as in linear regression) is present

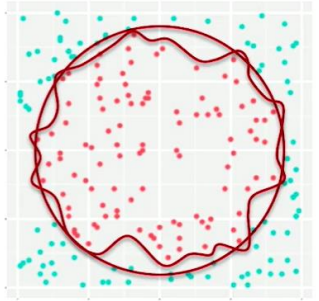




$|x| + |y| = c$



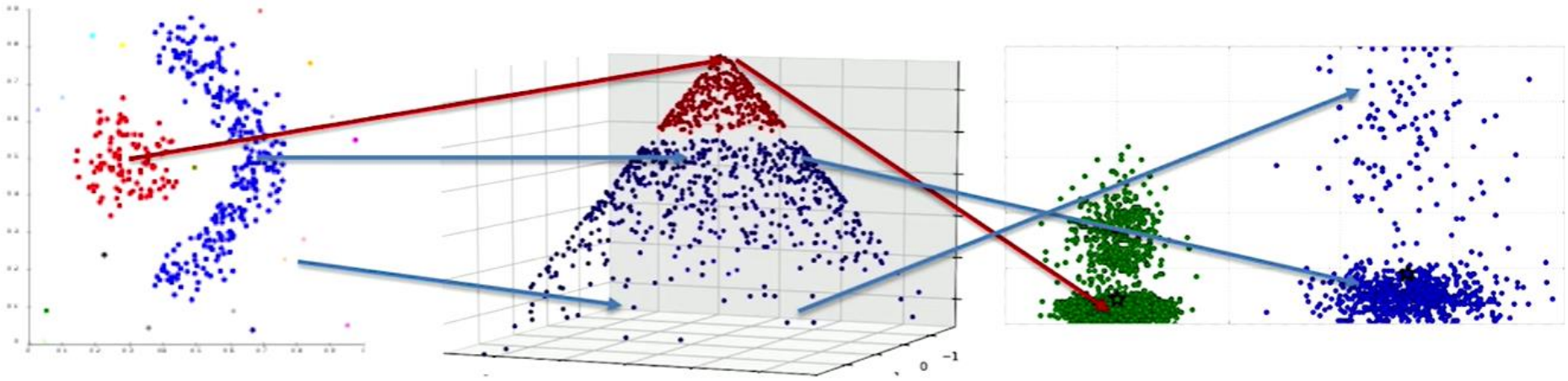
$x^2 + y^2 = c$



$x_1^a + x_2^b \ldots + x_n^p = c$

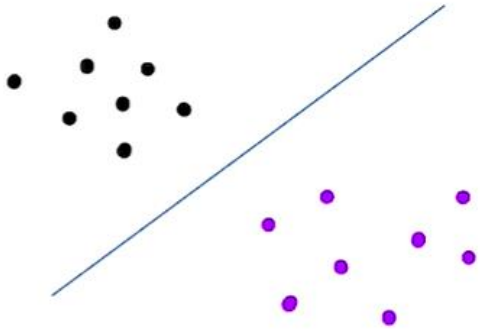
Graph as you need

- The decision boundary line type depends on the features and how you graph the data
  - Then your decision boundary equation will determine if a point is “more likely in group A or in group B”



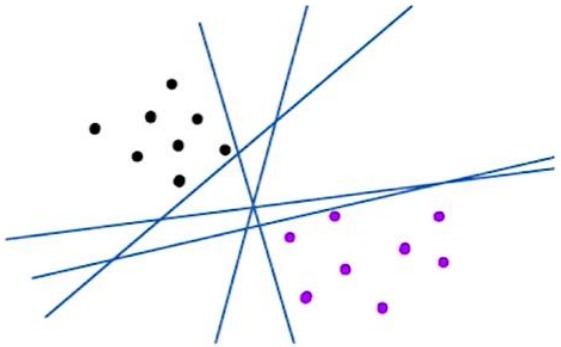
When Sigmoid Does not Work for you

- For many graphs, logistic regression is not the best choice
  - All you need is find the groups (and their members)
  - The graph structure may not work well for sigmoid... and you don't want to change the graph:
    - Data shows what you want
    - You have more than 2 groups

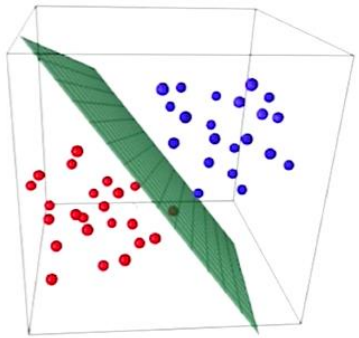


Looking for the Separation line

- There may be multiple possible separation lines
- Support Vector Machines (SVM) helps you find the ‘best’ line

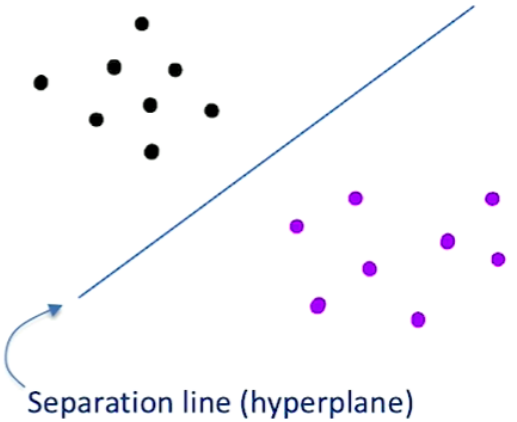


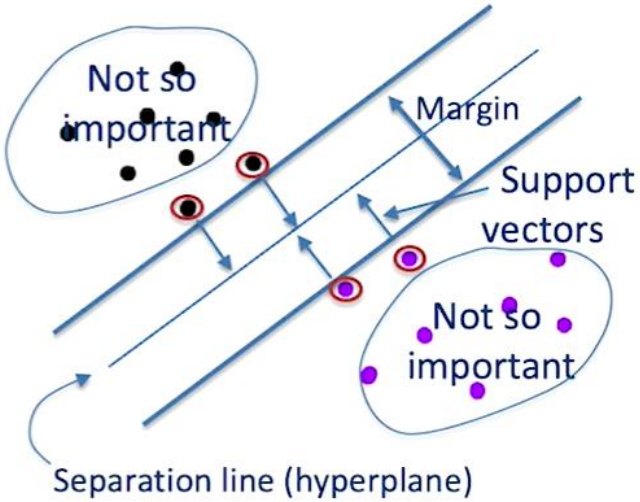
- If you have more than 2 dimensions, the ‘line’ becomes ‘a plane’, or ‘a hyperplane’
- The ‘best’ line is the one farthest from all points



SVM Terminology

- SVM helps you find the middle line
  - Once you determine the groups, consider the points at the edge
  - Then finding the line that is farthest from these support vectors is a ‘constrained optimization problem’
    - Farthest means maximizing the ‘margin’
  - SVM solves it with a formula called Lagrange multipliers



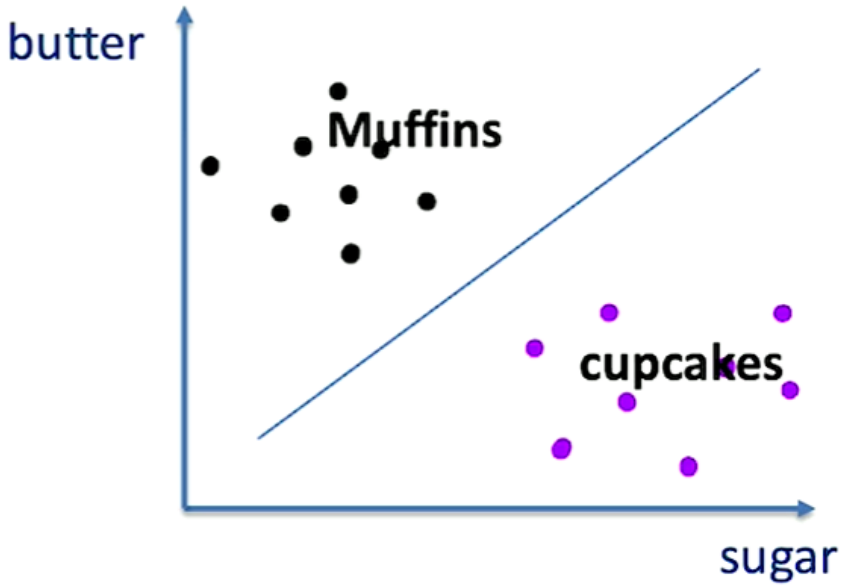


Finding Max Distance

- Remember that this is your training set
  - You know which point is in which group
- E.g muffins vs cupcake
  - Muffins: lots of flour, quite some milk, quite some sugar, low butter
  - Cupcake: quite some flour, some milk, some sugar and quite some butter

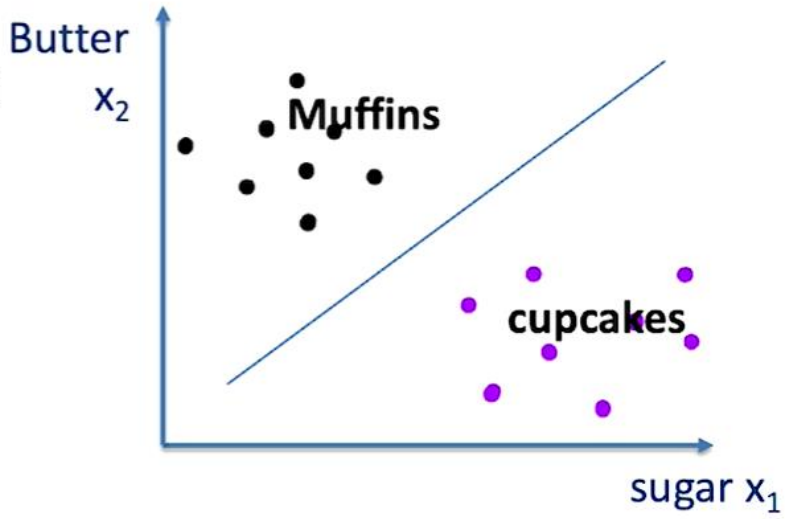
Finding Max Distance

- sugar = x_1, butter = x_2, flour = x_3, milk = x_4
- To simplify the graph, let's look at butter vs sugar:
  - Small x_1, large x_2 = muffin
  - Large x_1, small x_2 = cupcake
- Each dimension has a weight
  - High w_1, low w_2 -> cupcake
  - Low w_1, large w_2 -> muffin



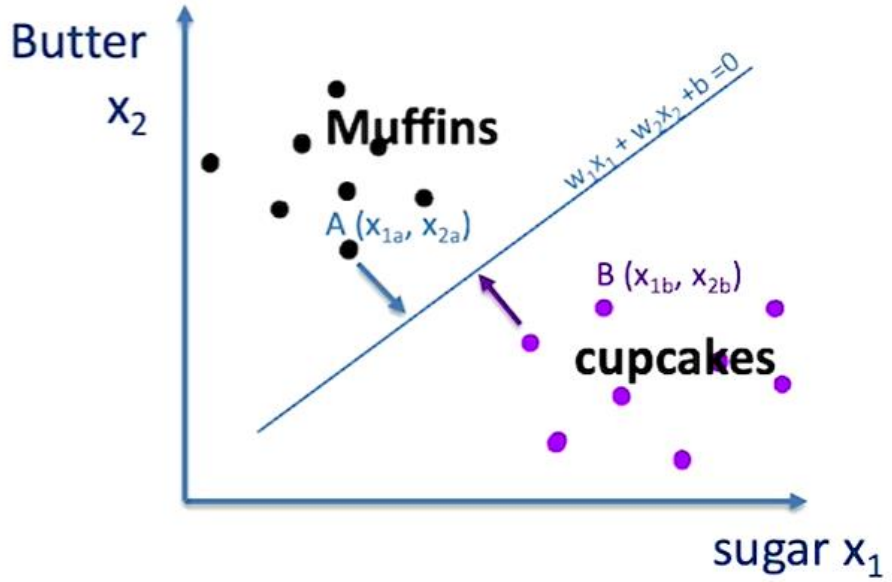
Finding Max Distance

- We use a similar idea of cost as in linear regression (with a variant)
- Start with a line:
  - It is slightly different from linear regression, instead of ax+b = y, you need wx+b = 0
    - W is your weight, here: $w_1x_1 + w_2x_2 + b = 0$
    - (the goal is to find the w's, positive result will mean one group, negative results will mean the other group)



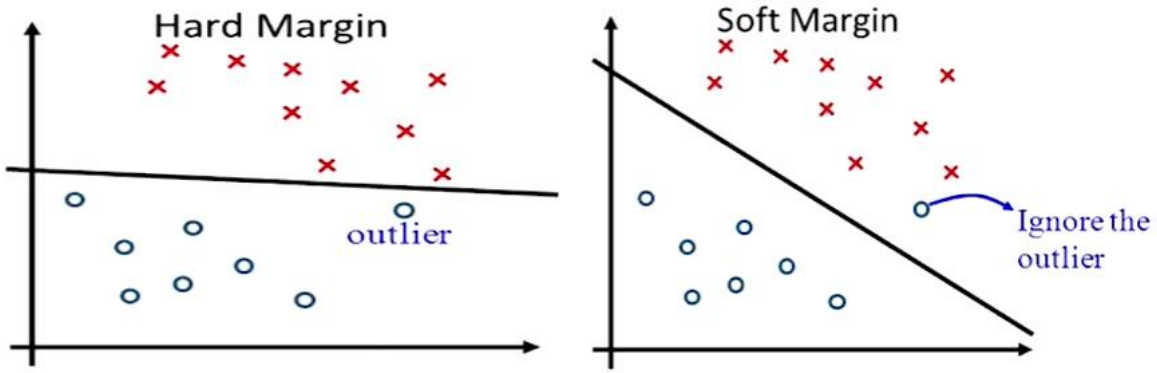
Finding Max Distance (Max Margin)

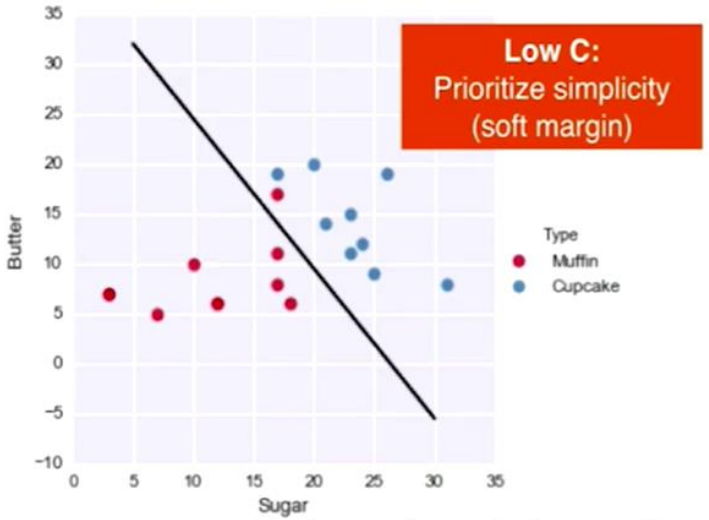
- Then for each point, compute your distance to the line:
  - A shortcut in linear algebra (vector dot product) helps us:
    - $d_A = w_1x_{1a} + w_2x_{2a} + b$
    - $d_B = w_1x_{1b} + w_2x_{2b} + b$
    - When the arrow goes one direction, result will be positive, negative in the other direction
  - Run the computation through your training set… and your goal is to find the maximum distance (max margin)

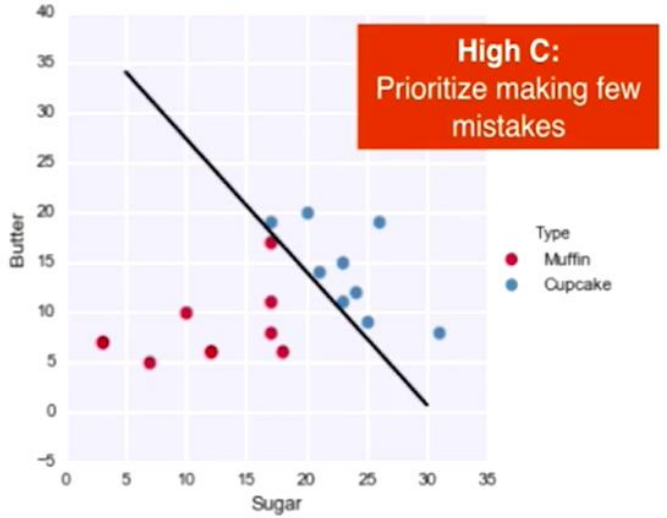


Soft Margin

- Most ‘modern’ SVM ML tools allow you to compute the margin… and be soft with it
  - Useful when your training set has ‘normal’ outliers
  - This ‘softness’ is often called the ‘C parameter’



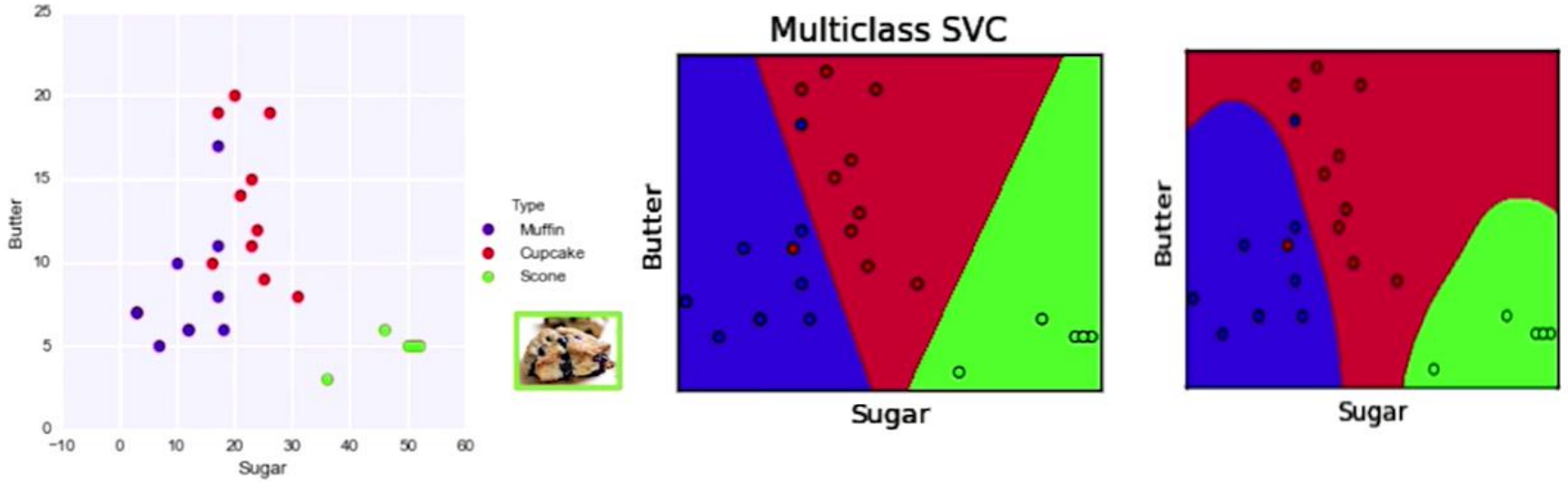




- Determining the right softness is still your job

C is also known as the regularization parameter that we saw in lecture 2.

Multiple Classes

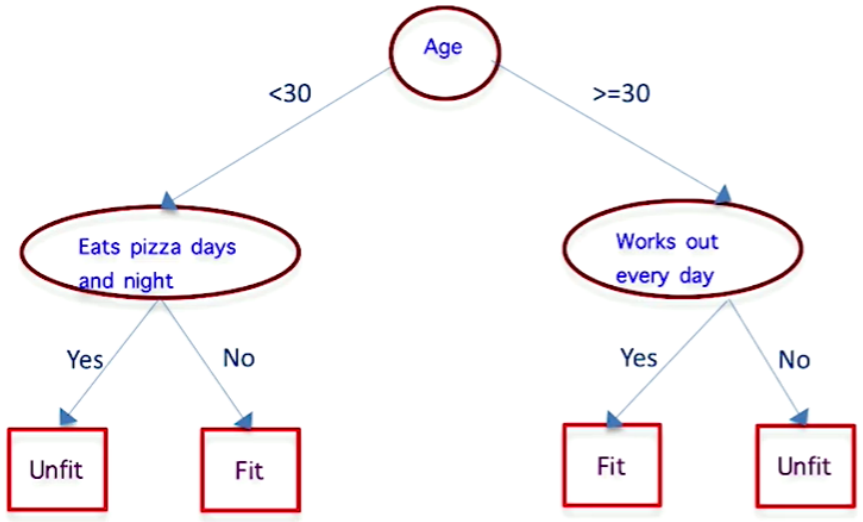


- You can determine more than one line
  - Process is fairly similar
  - And there are math projections to twist lines and dimensions

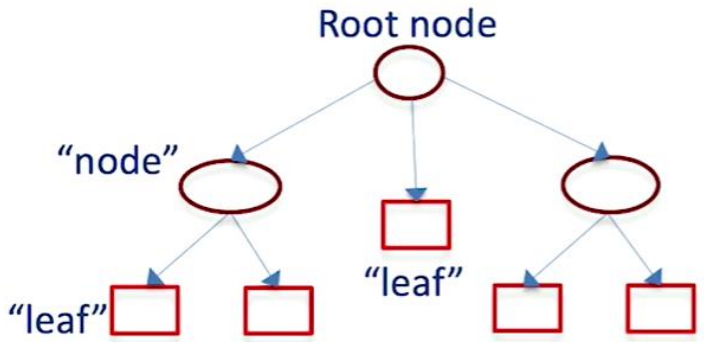
Exploring Random Forests

Decision Trees

- You probably know them… common on paper and in ML
  - Each step asks a question, classifies the answer then branches out depending on that answer
    - Answer can be number, yes/no (0/1), name/category.



- Decision trees can represent categories or numbers
- When data becomes large, ML can help build the tree structure (computes relationship between data points)
  - If variables are continuous numbers, it is a (linear) regression algorithm
  - If variables are categories, it is a classification algorithm



How to Grow a Tree

- Multiple ways
- Suppose you want to test potential of a customer buying a computer
  - Pick one variable (e.g. credit)
  - Then check if there is a good match

Fair rating	
Yes	No
5	2

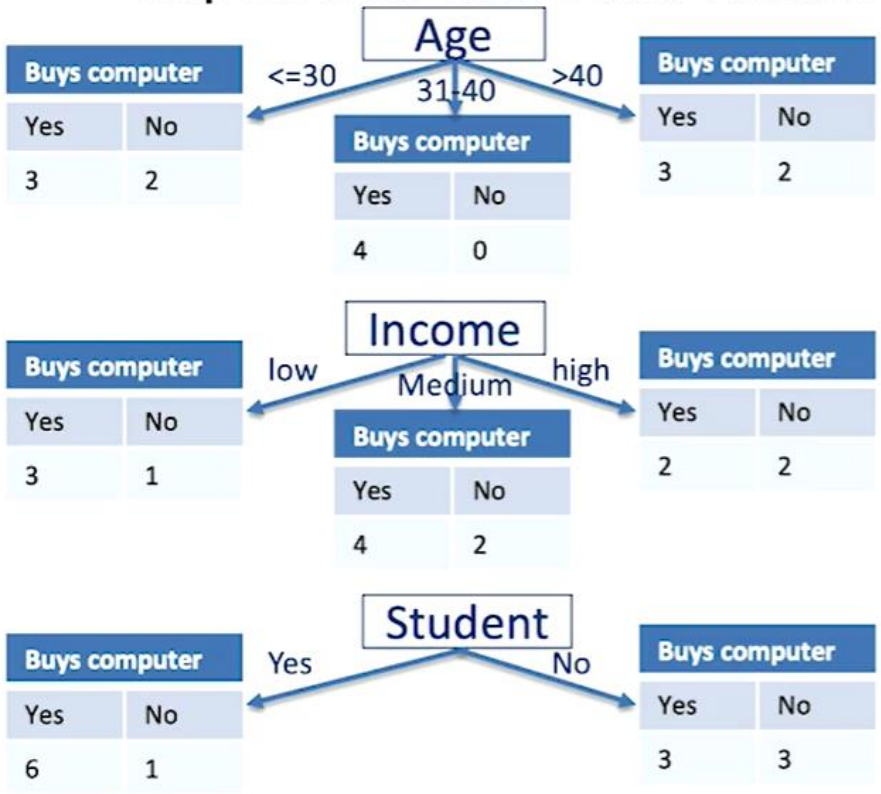
Excellent rating	
Yes	No
3	3

credit_rating	buys_computer
fair	no
excellent	no
fair	yes
fair	yes
fair	yes
excellent	no
excellent	yes
fair	no
fair	yes
fair	yes
excellent	yes
excellent	yes
fair	yes
excellent	no

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

How to Grow a Tree

- Repeat with each other variable



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

Impure Trees

- None of our parameters are a perfect match to predict 'buy computer'
  - They are called 'impure'
- Choosing the best of 'impure' can be done in multiple ways, a popular one is computing the Gini impurity
- It's the same concept of 'distance' or 'cost' we saw before, applied to probabilities

"Cost" ported to Probability

- Gini impurity = 1 – (probability of yes)^2 – (probability of no)^2

- Repeat for all leaves, then pick the "least impure" as your root node

Why Does Gini Work?

- Gini impurity = 1 – (probability of yes)^2 – (probability of no)^2

Buys computer	
Yes	No
6	1

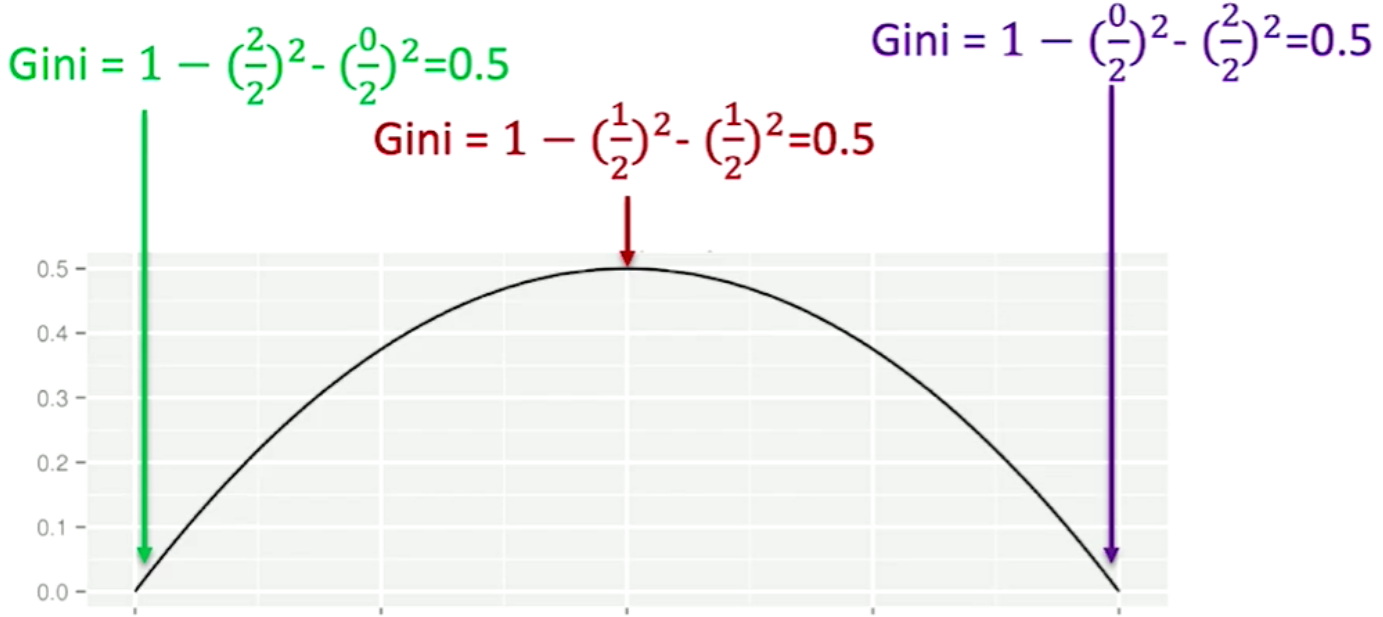
$$\text{Gini} = 1 - \left(\frac{6}{6+1}\right)^2 - \left(\frac{1}{6+1}\right)^2 = 0.245$$

- In a perfect ("pure") world, there is a perfect match between the leaf results and "buy computers":

Everyone buys computers	No one buys computers
1 – (probability of yes)^2 – (probability of no)^2	1 – (probability of yes)^2 – (probability of no)^2
1 – this is 1 – this is 0	1 – this is 0 – this is 1
1 – 1 – 0 = 0 (pure)	1 – 0 – 1 = 0 (pure)

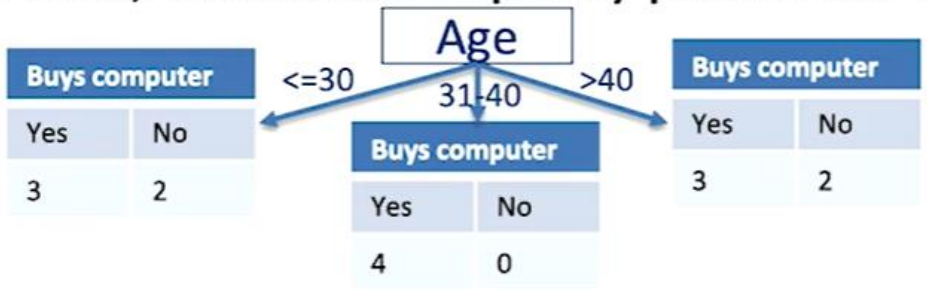
The Gini Impurity Graph

- Gini impurity measures how random your match is
  - Very impure = completely random -> works 50% of the time, no more



Completing the Tree

- In our case, the lowest impurity parameter ends up being 'Age"



- Once you have a root node, take each branch separately and re-apply the process:

<=30 = 5 | 31-40 = 4 | > 40 = 5

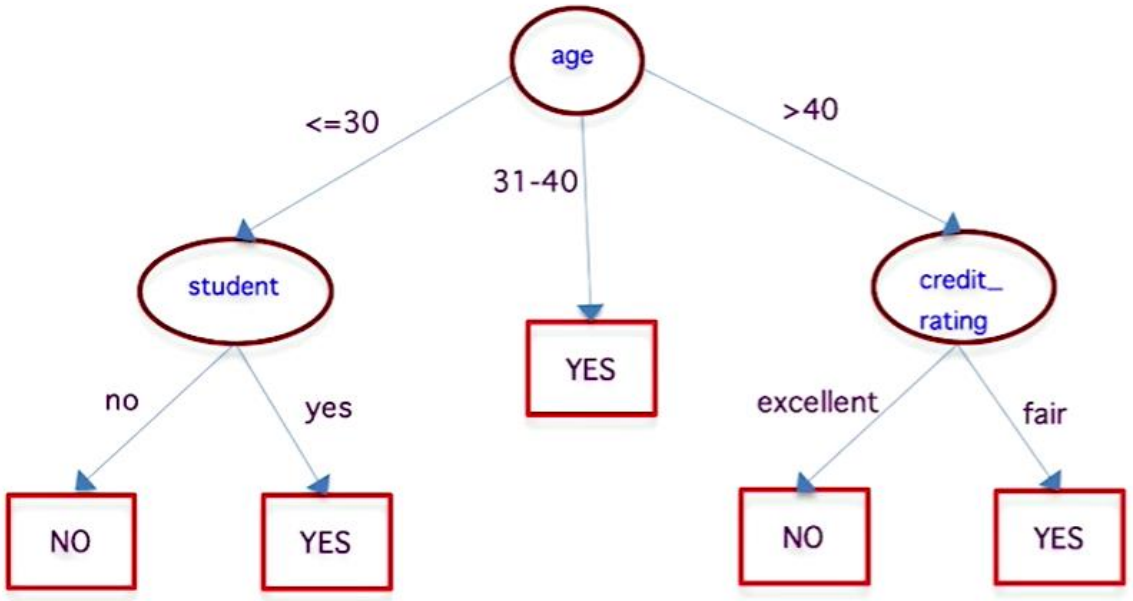
Out of these, student/non student? Income? Credit?

Student(1)	high (2)	Excellent (2)
Non-std(4)	Medium (2)	Fair (3)
	Low (1)	

Then repeat Gini impurity to build your next node

Completing the Tree

- You grew your first tree!



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

Random Forests

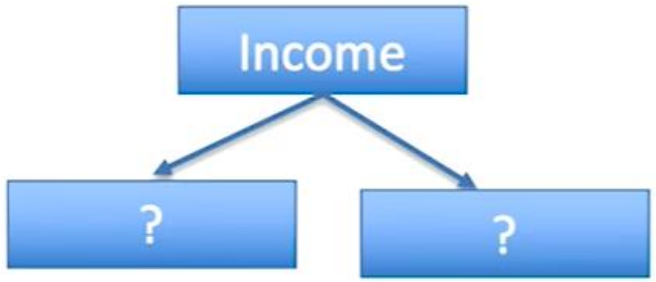
- Forests are made of trees…
- Trees are limited, in that they work well with the data used to create them, but adapt poorly to other data sets
- The goal of the random forest is to create multiple trees, that together will get better accuracy
  - Better adaptation to new data
  - Better prediction
- To create a random forest, create a bootstrap data set
  - Take random variable subset (same or smaller variable size)
  - Take sample data (can take the same sample more than once)

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

Appears twice

>40	medium	no	fair	yes
<=30	high	no	excellent	no
31-40	medium	no	excellent	yes
<=30	high	no	excellent	no

Random Forests Principles



- Ignore income for the rest of the work (already used)
- Then select 2 new random variables as candidate for each node
  - Then build the tree that way

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

>40	medium	no	fair	yes
<=30	high	no	excellent	no
31-40	medium	no	excellent	yes
<=30	high	no	excellent	no

Random Forests Principles

- Then, build trees
  - But only take a (random) subset of variables (columns) at each step

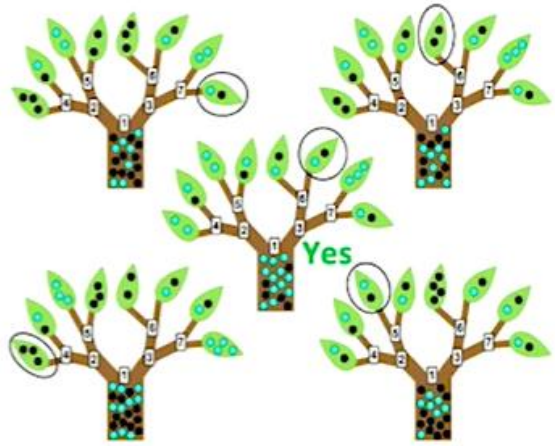
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31-40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31-40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31-40	medium	no	excellent	yes
31-40	high	yes	fair	yes
>40	medium	no	excellent	no

Suppose ‘income’ worked to create the root node

>40	medium	no	fair	yes
<=30	high	no	excellent	no
31-40	medium	no	excellent	yes
<=30	high	no	excellent	no

Building More Trees

- Once you have a tree, restart from the beginning!
  - Take your subset
  - Pick 2 variable randomly
  - Build the first node
  - Repeat down the tree
- In the end, you build hundreds of trees



Testing your Forest

- Creating a **B**ootstrap and using **agg**regates to build the forest is called **bagging**
- Once the forest is built, take data in your training set that you did not use for the bootstrap, and test it against the forest
  - Called 'out of bag data'
  - This allows you to test the efficiency of your forest
    - As you know your 'buy computer' value, you can test the percentage of trees that get the answer 'right' -> this is the forest efficiency

What to do with all these trees

- When new data comes in (a new potential computer buyer in our example):
  - Collect the data: | 31-40 | medium | no | fair |
  - Then run the data through each tree:
    - Buy computers?
  - The max votes wins!
    - No = 1 , Yes = 4 -> **Yes**