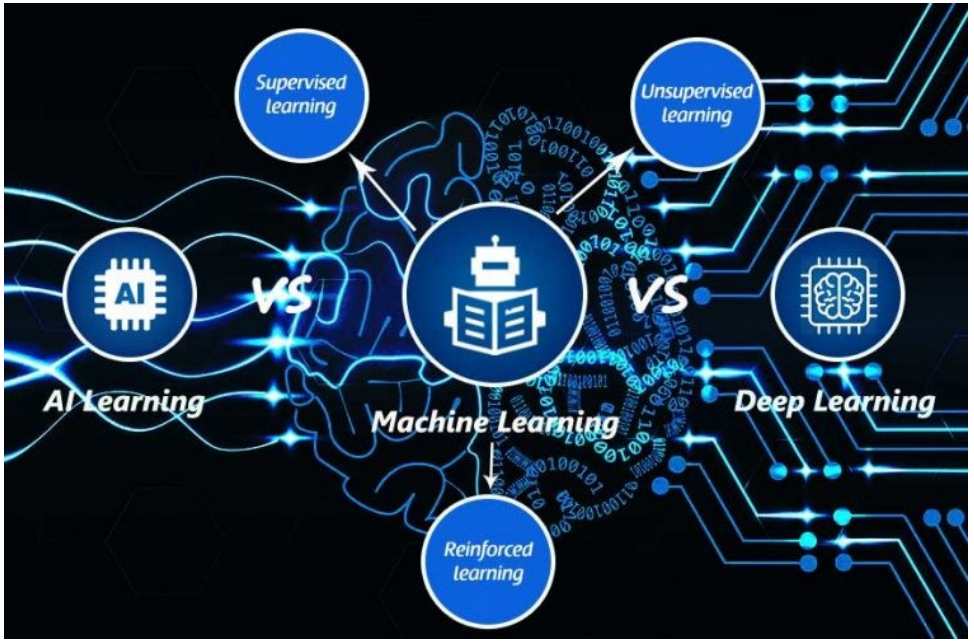
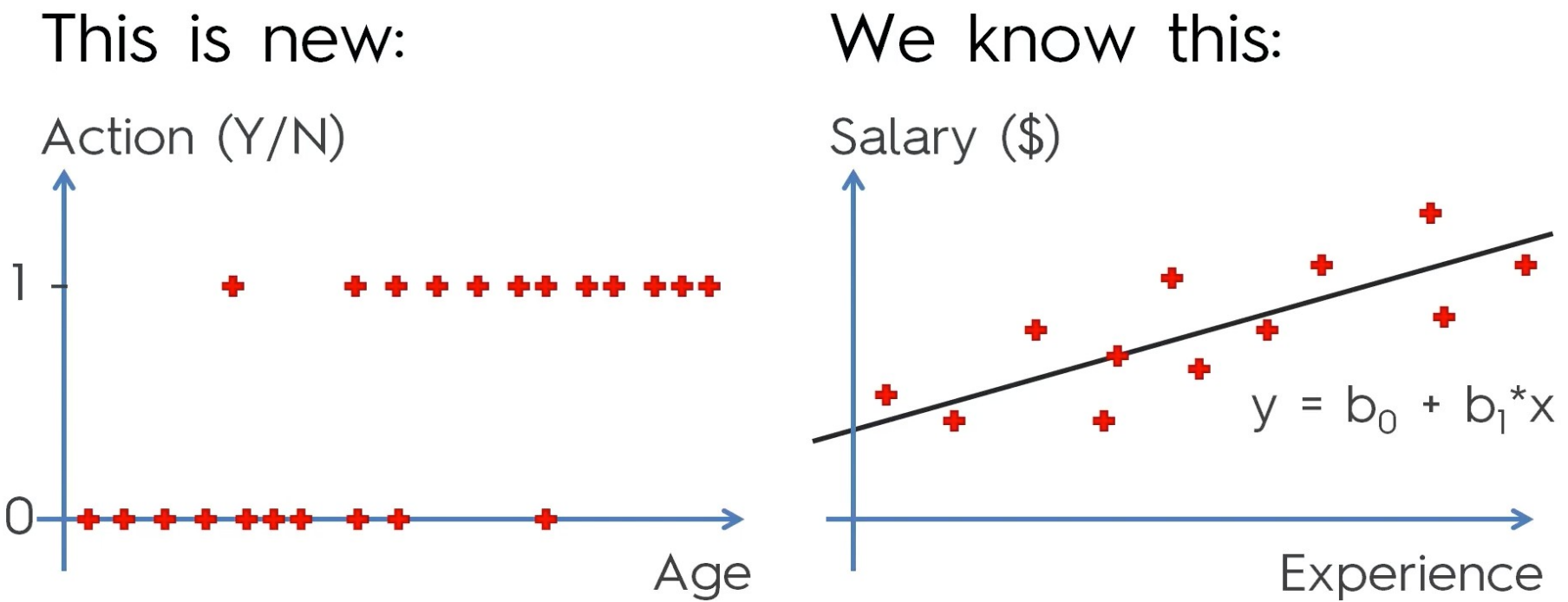


İTÜ
Machine Learning Lessons

Lesson2

CLASSIFICATION

1) Logistic Regression



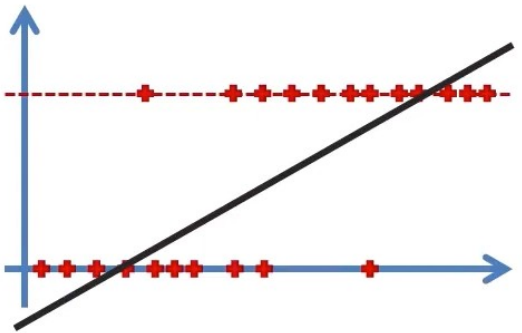
LOGISTIC REGRESSION

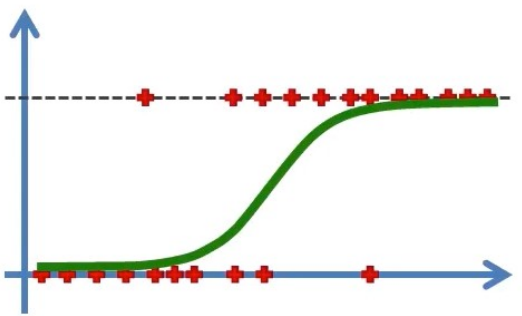
$$y = b_0 + b_1 * x$$

Sigmoid Function

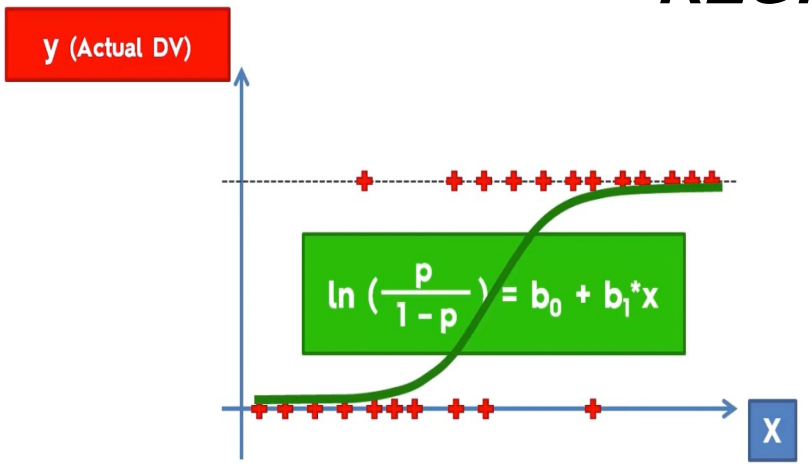
$$p = \frac{1}{1 + e^{-y}}$$

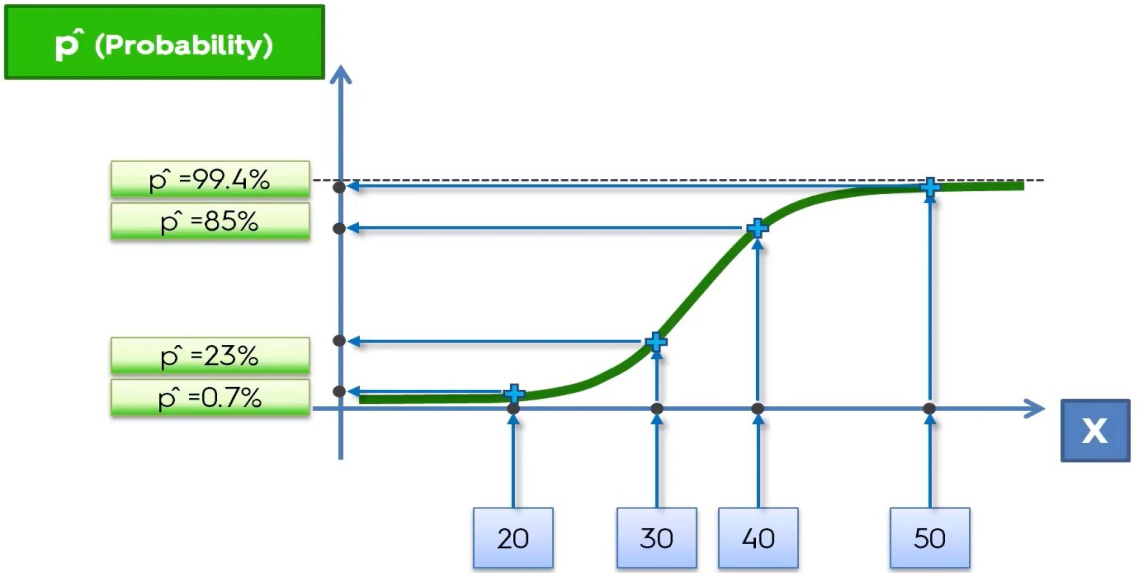
$$\ln\left(\frac{p}{1-p}\right) = b_0 + b_1 * x$$



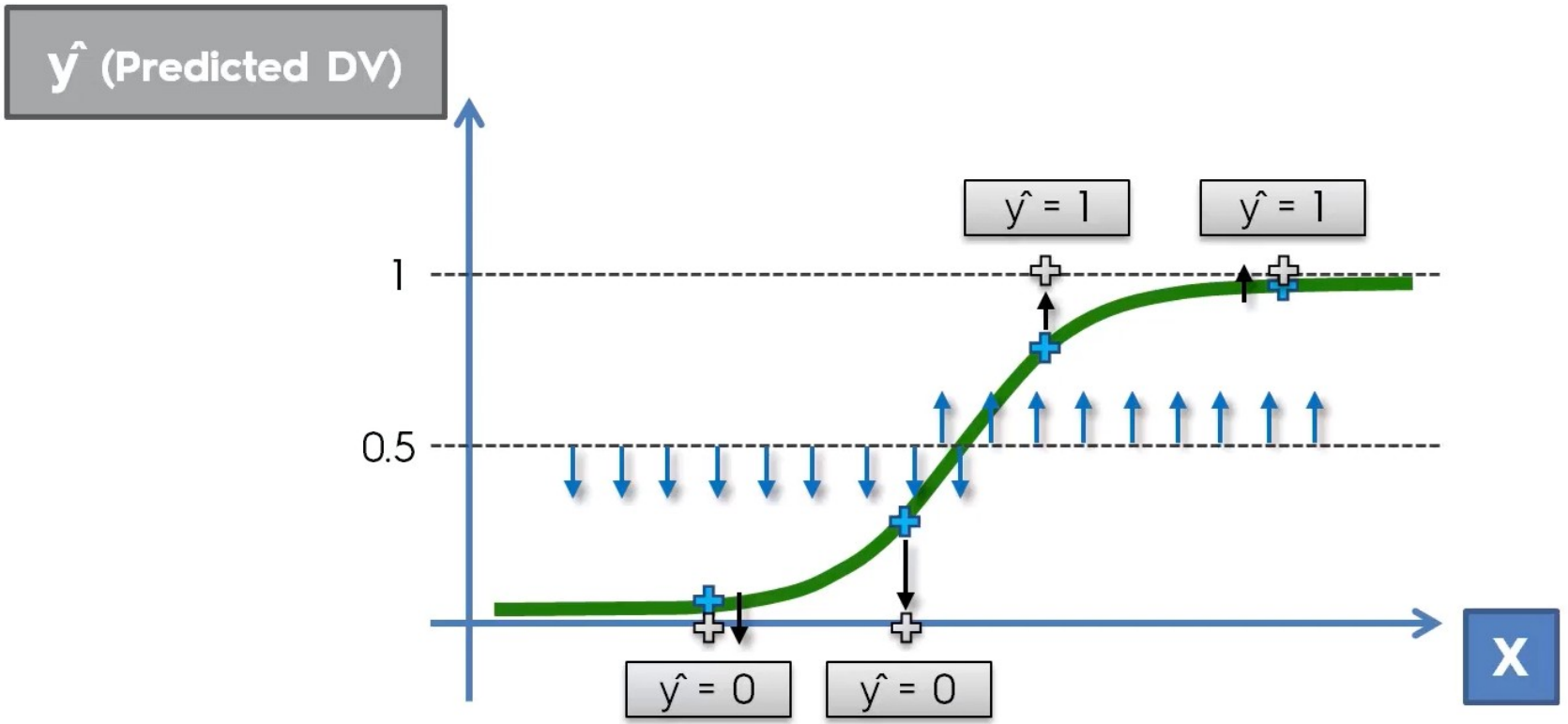


LOGISTIC REGRESSION

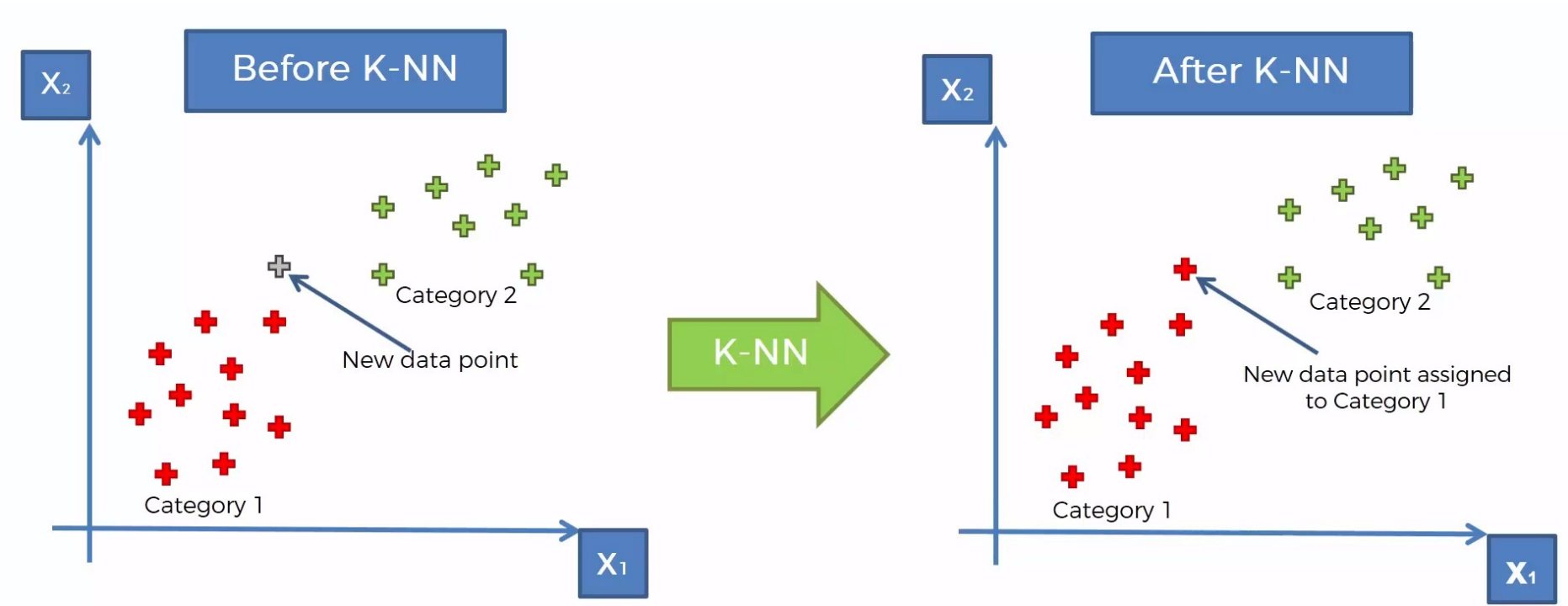




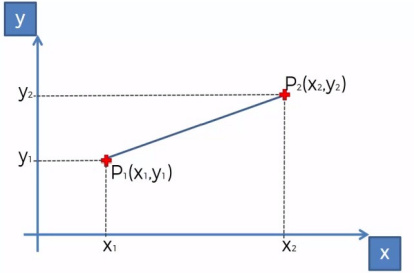
LOGISTIC REGRESSION



K-NEAREST NEIGHBOR (KNN)



K-NEAREST NEIGHBOR (KNN)



Euclidean Distance between P_1 and $P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

STEP 1: Choose the number K of neighbors

STEP 2: Take the K nearest neighbors of the new data point, according to the Euclidean distance

STEP 3: Among these K neighbors, count the number of data points in each category

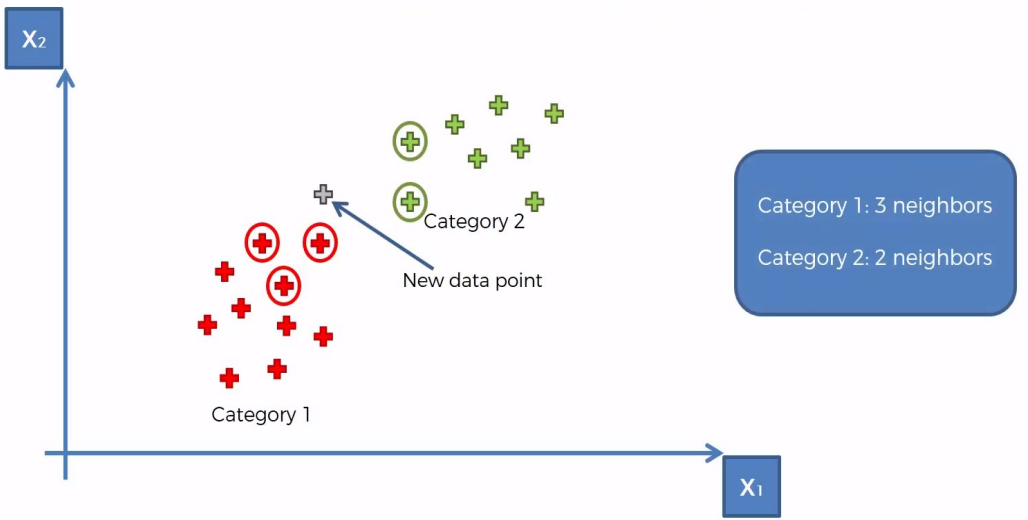
STEP 4: Assign the new data point to the category where you counted the most neighbors

Your Model is Ready

K-NEAREST NEIGHBOR (KNN)

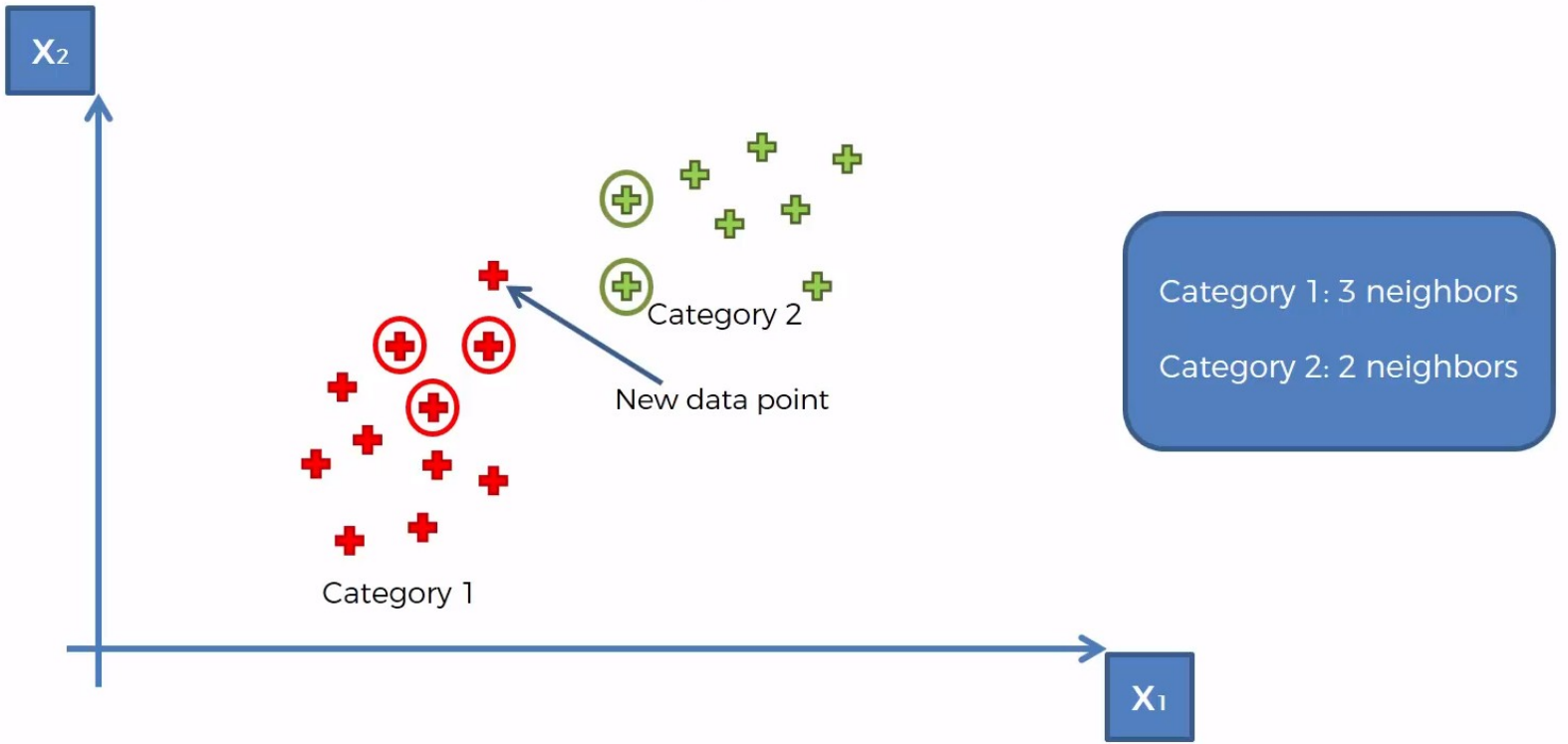
STEP 2: Take the K = 5 nearest neighbors of the new data point, according to the Euclidean distance

STEP 3: Among these K neighbors, count the number of data points in each category

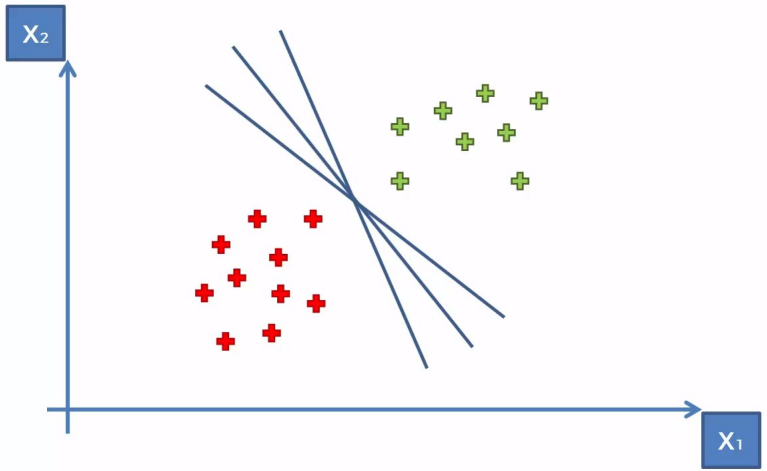


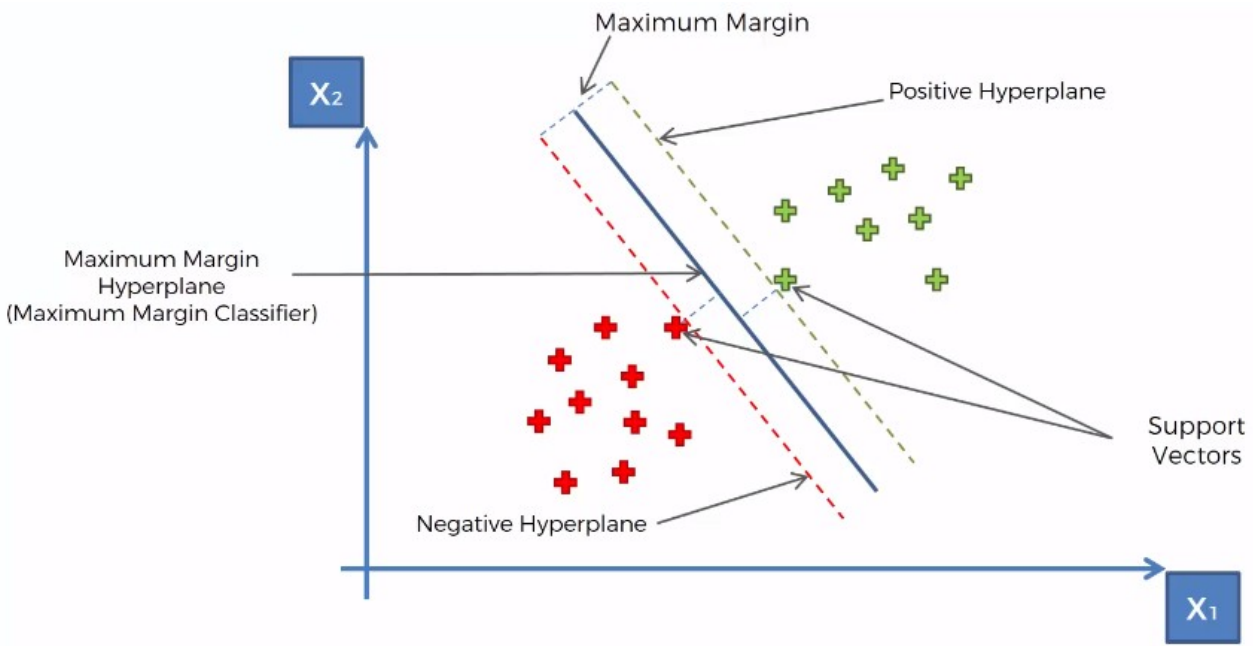
K-NEAREST NEIGHBOR (KNN)

STEP 4: Assign the new data point to the category where you counted the most neighbors

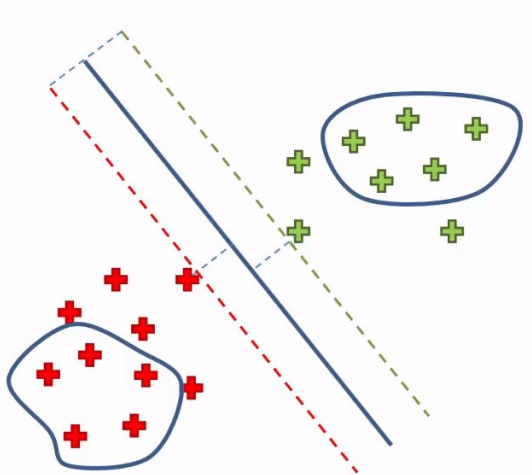


Support Vector Machine (SVM)

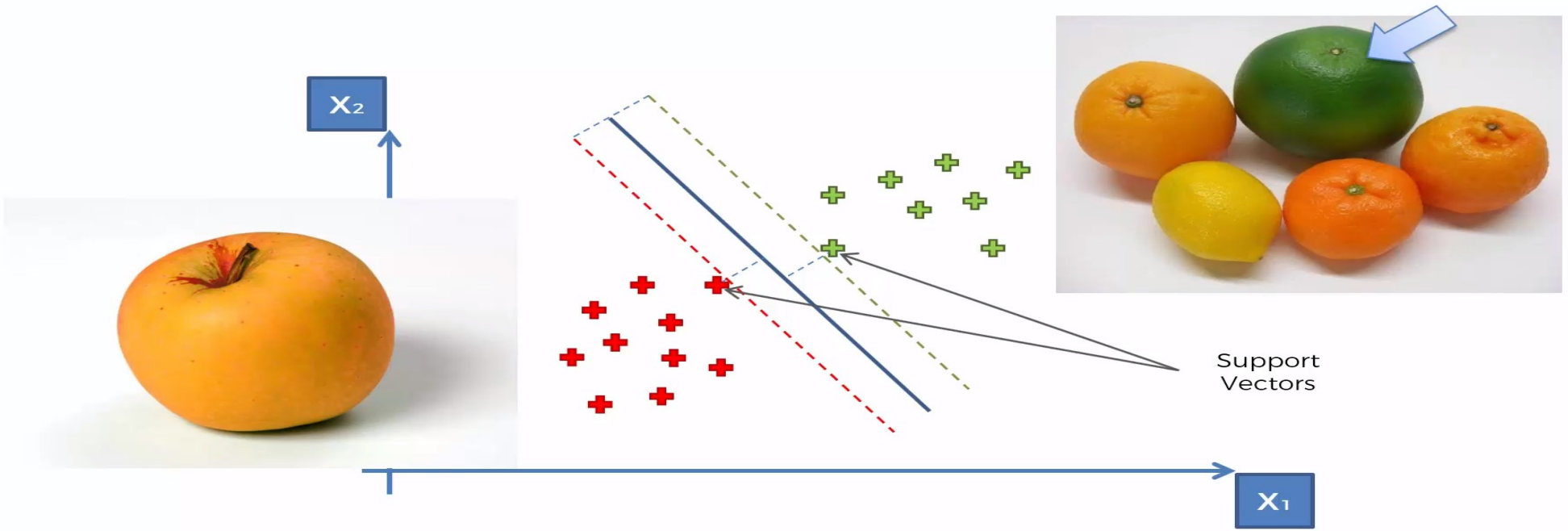




What's So Special About SVMs?

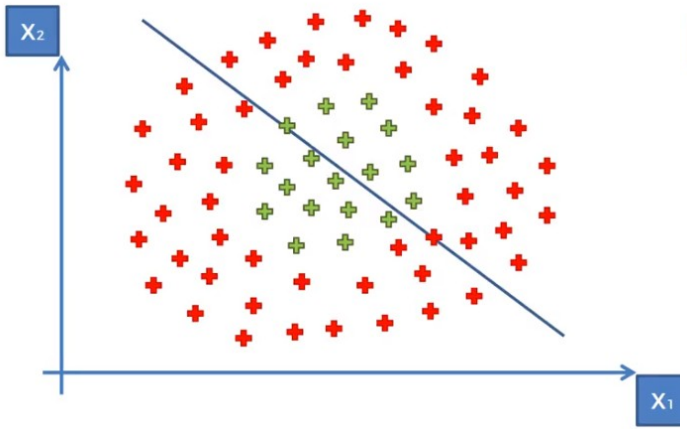


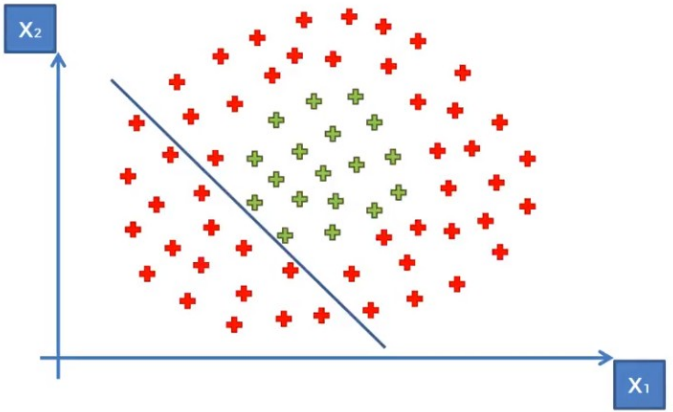
Support Vector Machine (SVM)

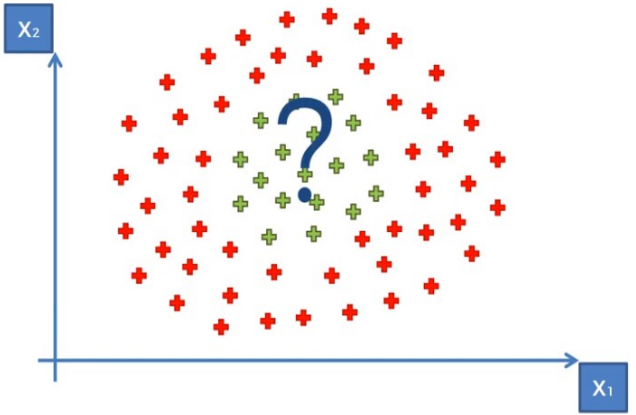


Support vectors are the examples of extremely ordinary part of their classes. In the other words, support vectors are look like an apple orange and an apple look like an orange

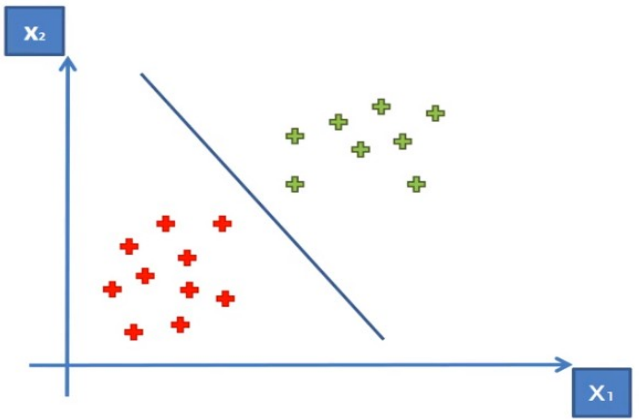
KERNEL SVM



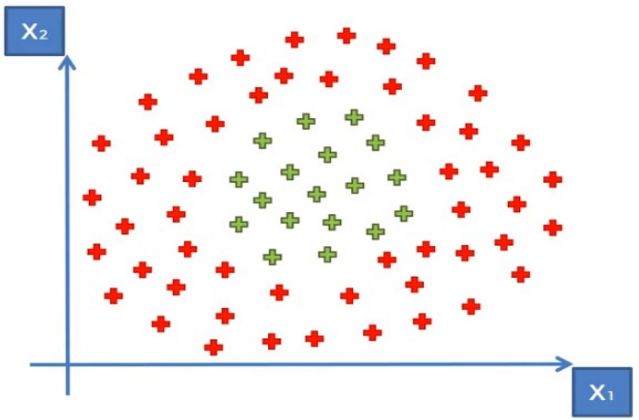




Linearly Separable

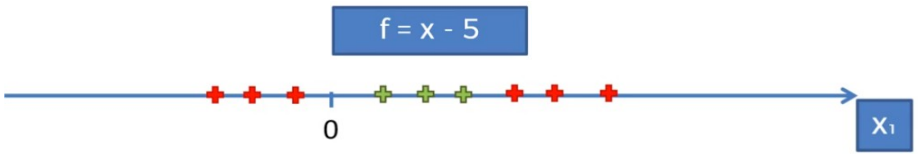


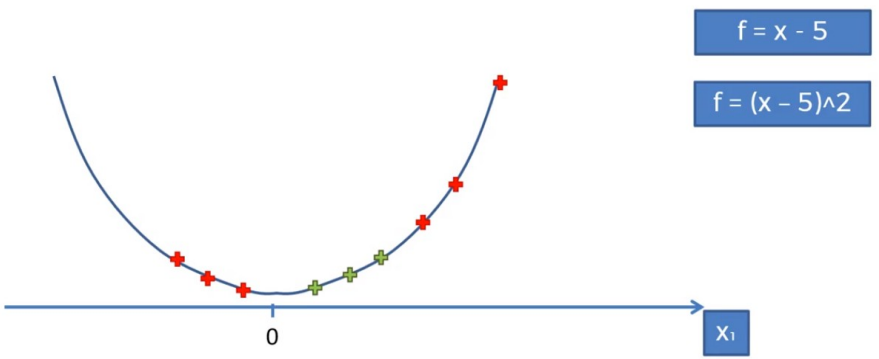
Not Linearly Separable

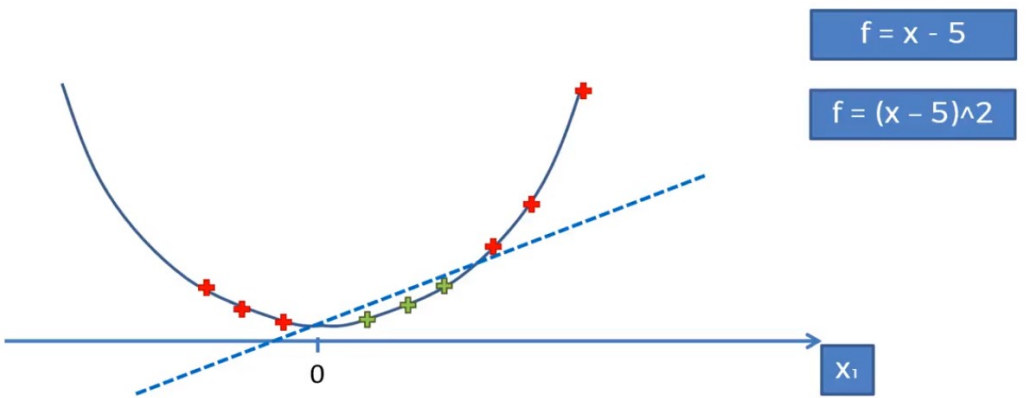


KERNEL SVM

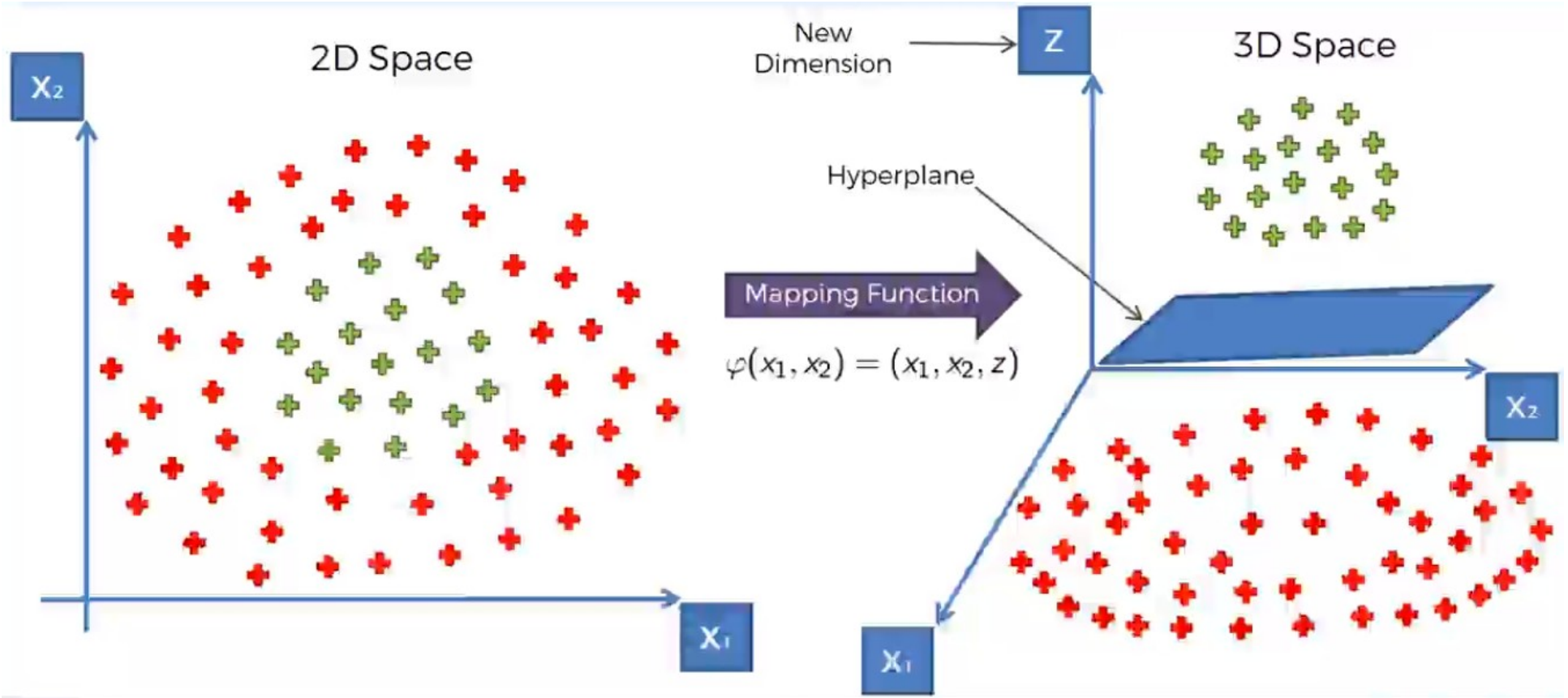
Mapping to Higher Dimension



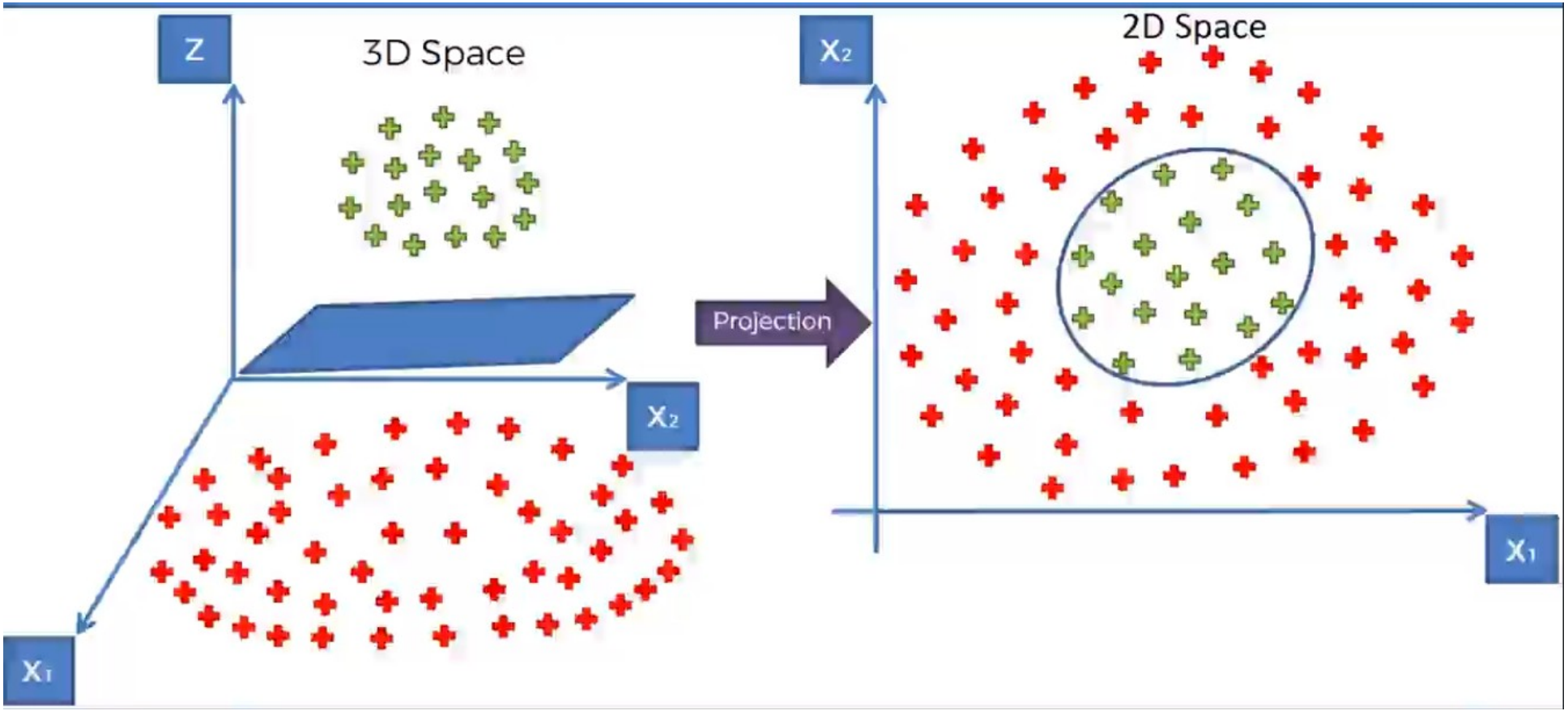




KERNEL SVM


2D Space
X2
X1
New Dimension
Z
3D Space
Hyperplane
Mapping Function
$\varphi(x_1, x_2) = (x_1, x_2, z)$
X2
X1

KERNEL SVM



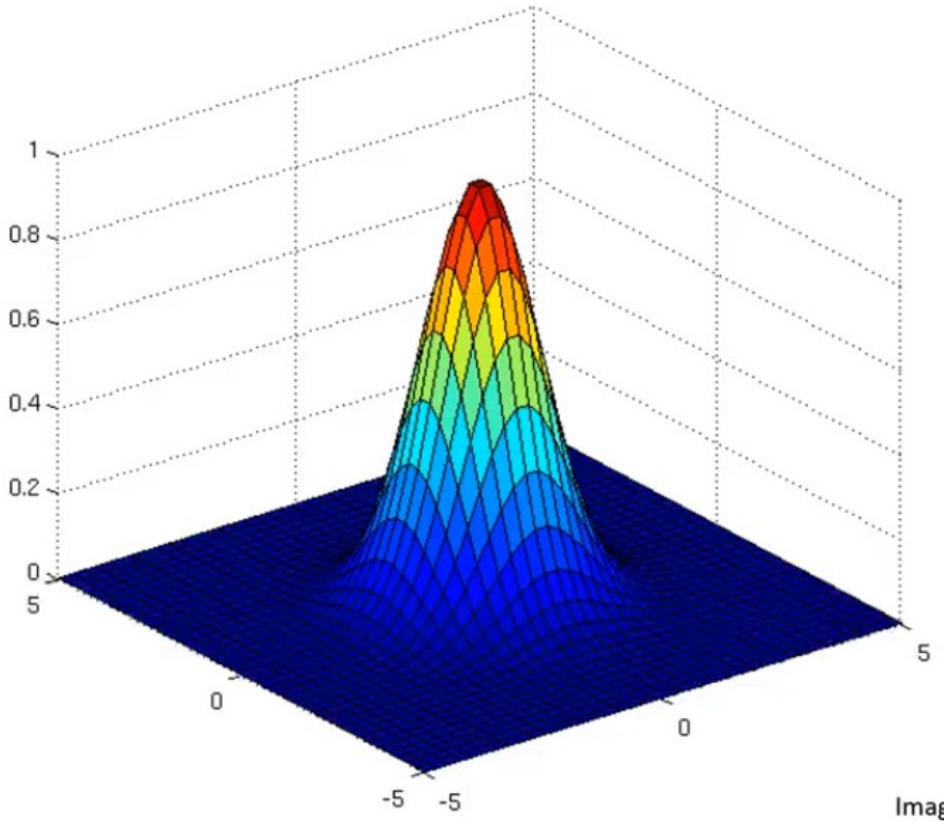
KERNEL SVM

But there's a problem with this algorithm and the problem is that mapping to a higher dimensional space can be highly compute intensive so it might require a lot of computation and a lot of processing power

In that reason and we're going to explore something else we're going explore a different approach which is called In mathematics the kernel trick.

Kernel Trick

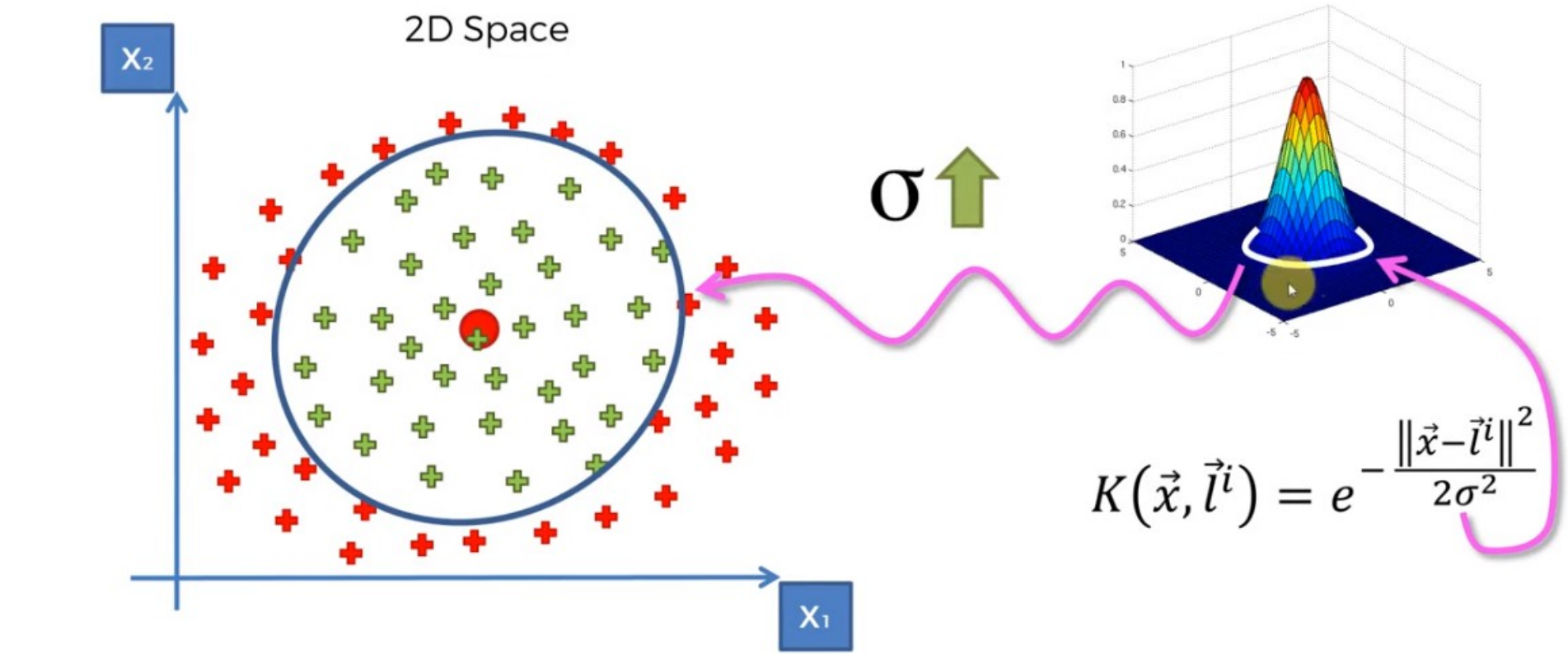
Gaussian RBF Kernel



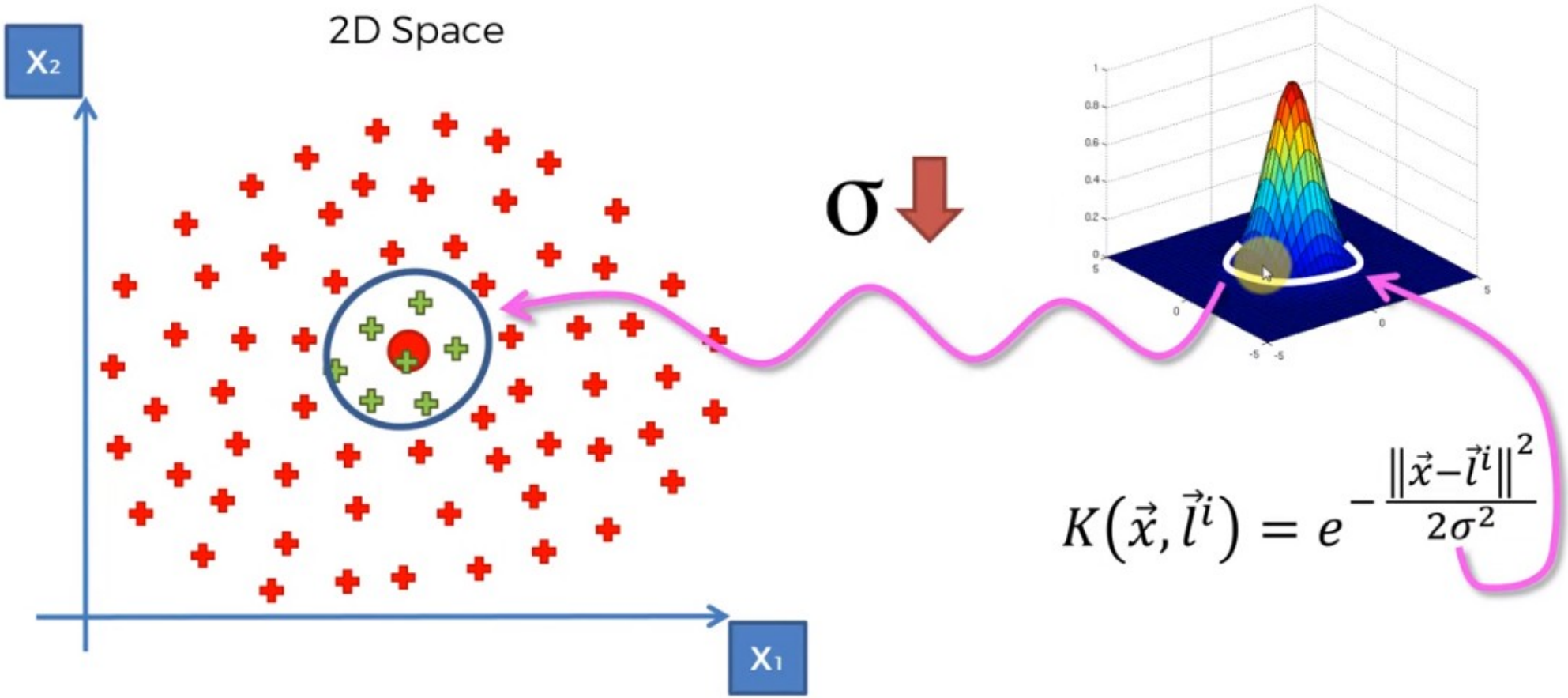
$$K\left(\vec{x}, \vec{l}^{i}\right) = e^{-\frac{\left\|\vec{x}-\vec{l}^{i}\right\|^{2}}{2\sigma^{2}}}$$

Image source: http://www.cs.toronto.edu/~duvenaud/cookbook/index.html

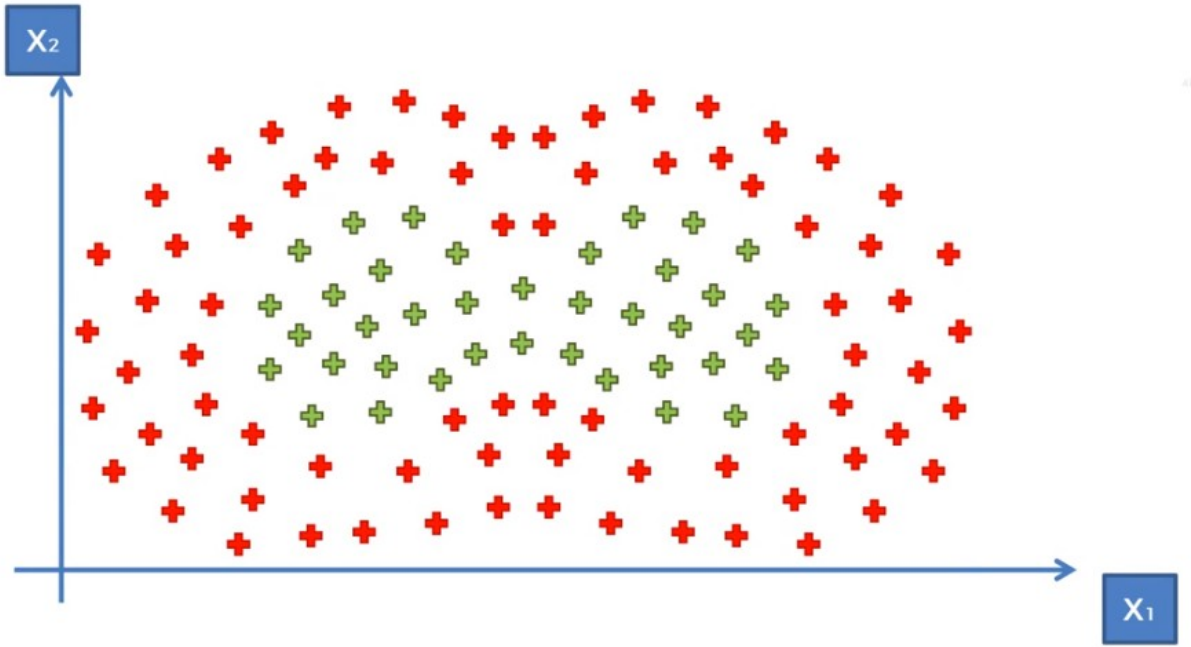
Kernel Trick



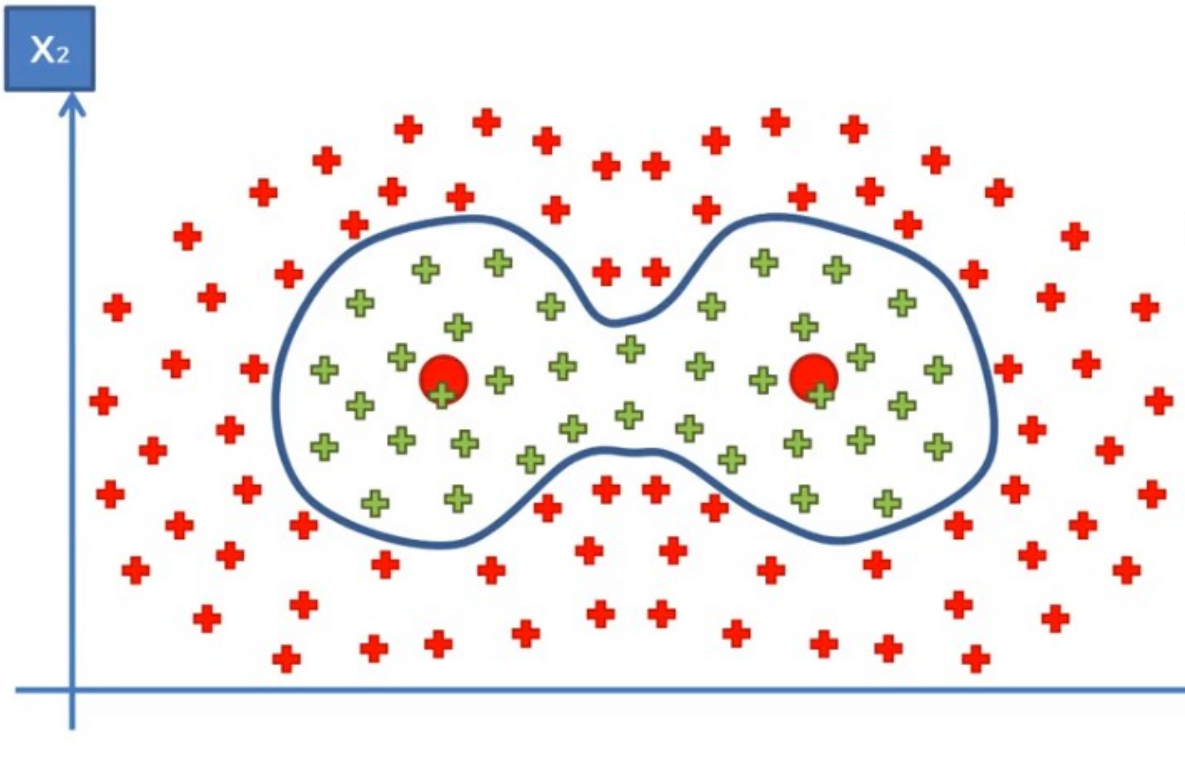
Kernel Trick



Kernel Trick



Kernel Trick



$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2)$$

(Simplified Formula)

Green when:

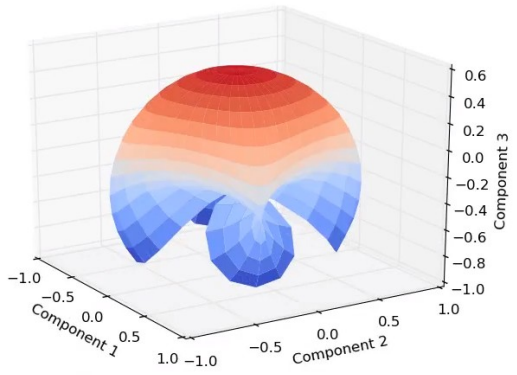
$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) > 0$$

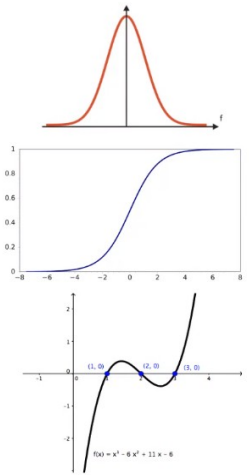
Red when:

$$K(\vec{x}, \vec{l}^1) + K(\vec{x}, \vec{l}^2) = 0$$

Kernel Types

Sine-Squared Kernel





Gaussian RBF Kernel

$$K\left(\vec{x},\vec{l}^{i}\right)=e^{-\frac{\left\|\vec{x}-\vec{l}^{i}\right\|^{2}}{2\sigma^{2}}}$$

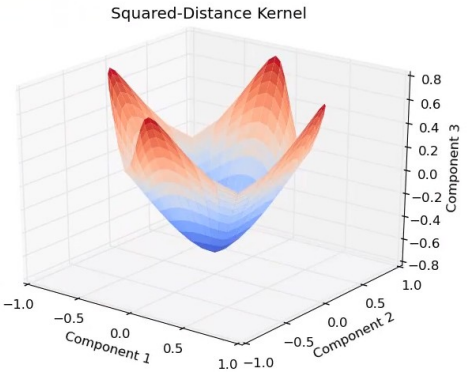
Sigmoid Kernel

$$K(X,Y)=\tanh(\gamma\cdot X^{T}Y+r)$$

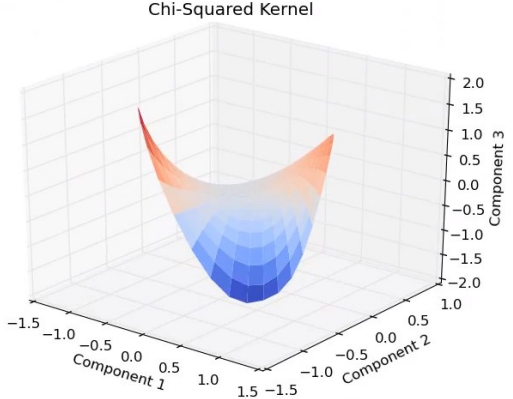
Polynomial Kernel

$$K(X,Y)=(\gamma\cdot X^{T}Y+r)^{d},\gamma>0$$

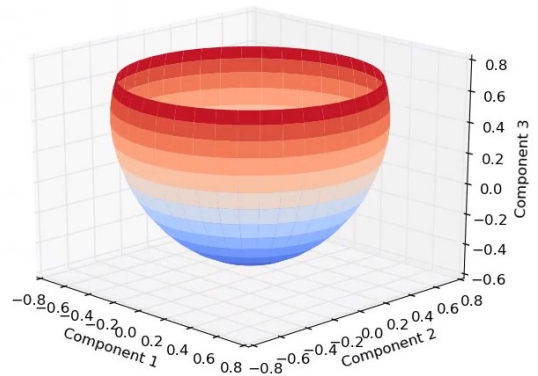
Squared-Distance Kernel



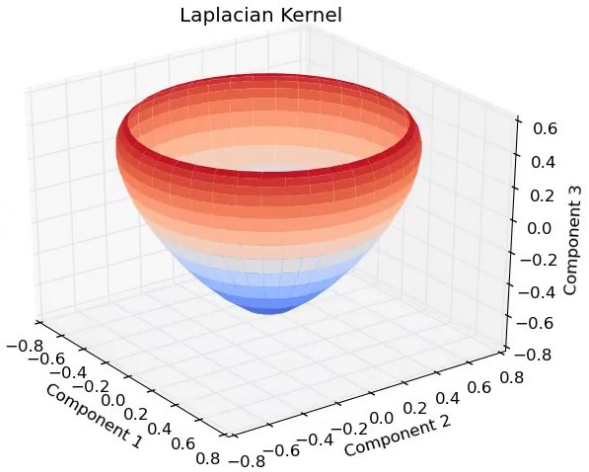
Chi-Squared Kernel

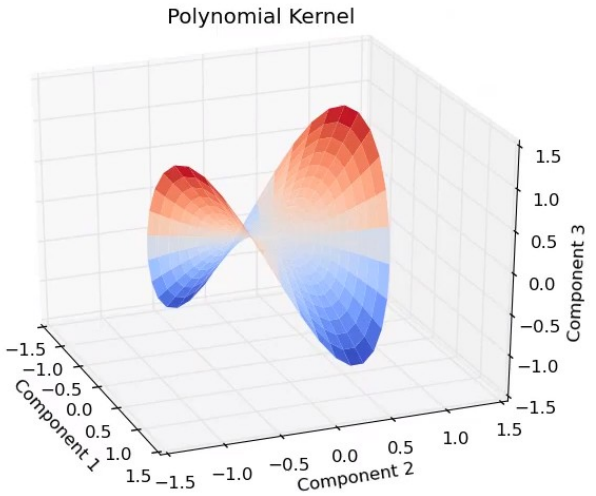


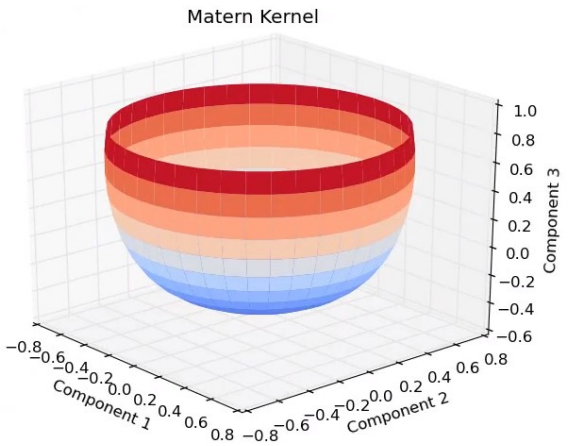
Gaussian Kernel

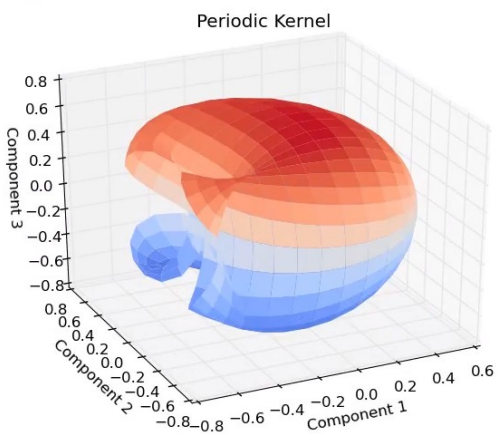


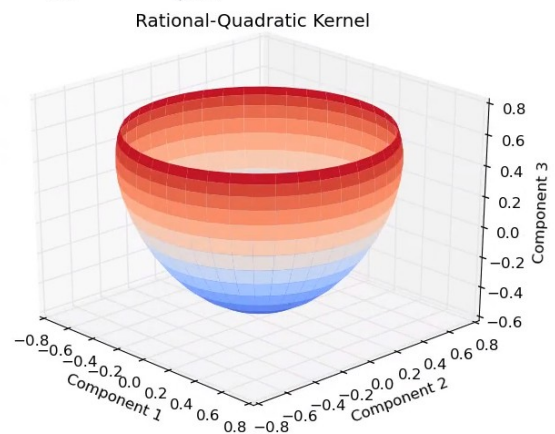
Kernel Types

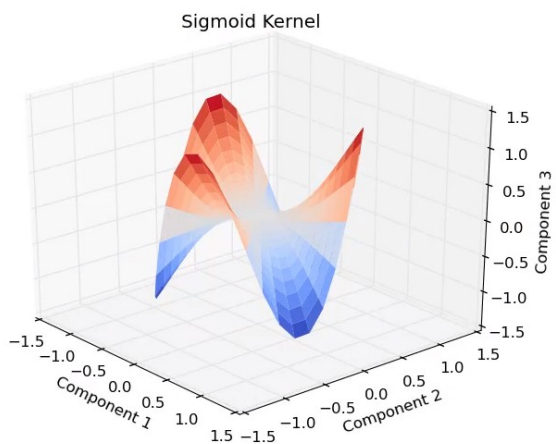




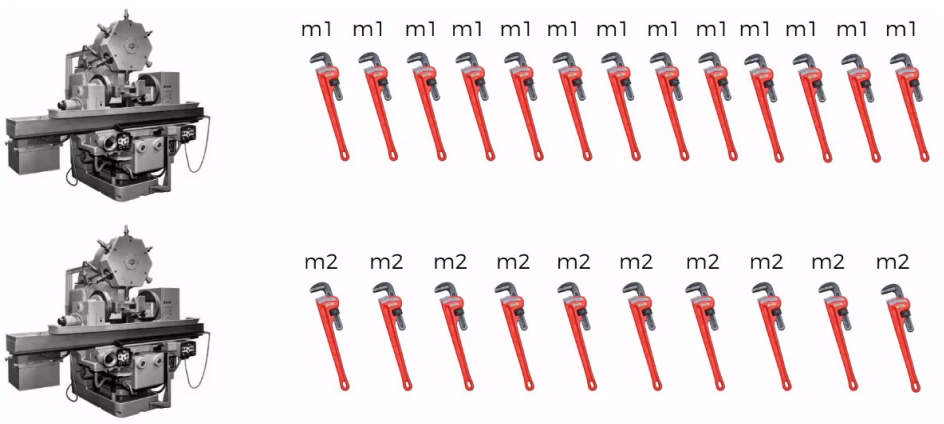




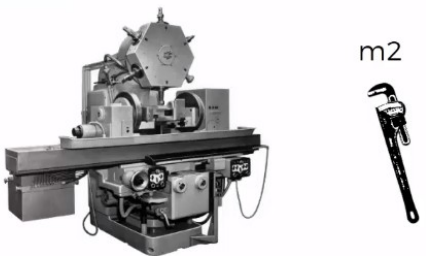




Naive Bayes Classification



What's the probability?

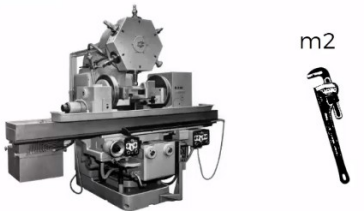


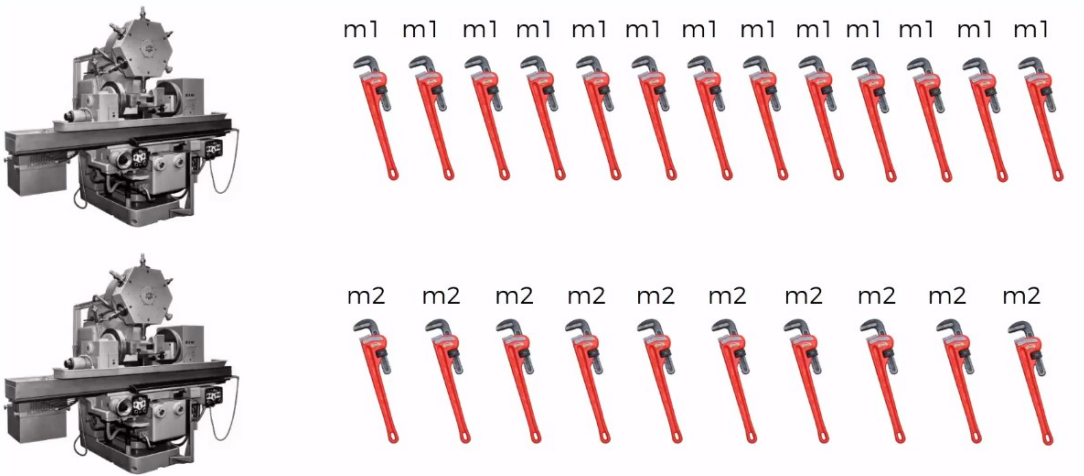
Naive Bayes Classification

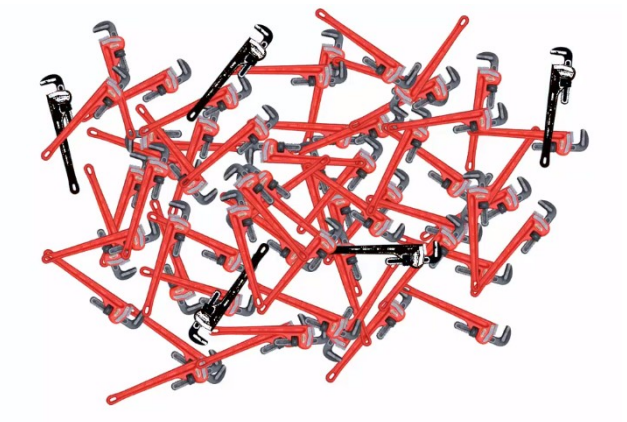
Bayes Theorem

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

What's the probability?







Naive Bayes Classification

Mach1: 30 wrenches / hr
Mach2: 20 wrenches / hr

-> P(Mach1) = 30/50 = 0.6
-> P(Mach2) = 20/50 = 0.4

Out of all produced parts:
We can SEE that 1% are defective

-> P(Defect) = 1%

Out of all defective parts:
We can SEE that 50% came from mach1
And 50% came from mach2

-> P(Mach1 | Defect) = 50%
-> P(Mach2 | Defect) = 50%

Question:
What is the probability that a part produced by mach2 is defective = ?

-> P(Defect | Mach2) = ?

Naive Bayes Classification

Mach1: 30 wrenches / hr
Mach2: 20 wrenches / hr
Out of all produced parts:
We can SEE that 1% are defective
Out of all defective parts:
We can SEE that 50% came from mach1
And 50% came from mach2
Question:
What is the probability that a part produced by mach2 is defective = ?

-> P(Mach2) = 20/50 = 0.4
-> P(Defect) = 1%
-> P(Mach2 | Defect) = 50%
-> P(Defect | Mach2) = ?

$$P(\text{Defect} \mid \text{Mach2}) = \frac{P(\text{Mach2} \mid \text{Defect}) * P(\text{Defect})}{P(\text{Mach2})}$$

$$P(\text{Defect} \mid \text{Mach2}) = \frac{0.5 * 0.01}{0.4} = 0.0125$$

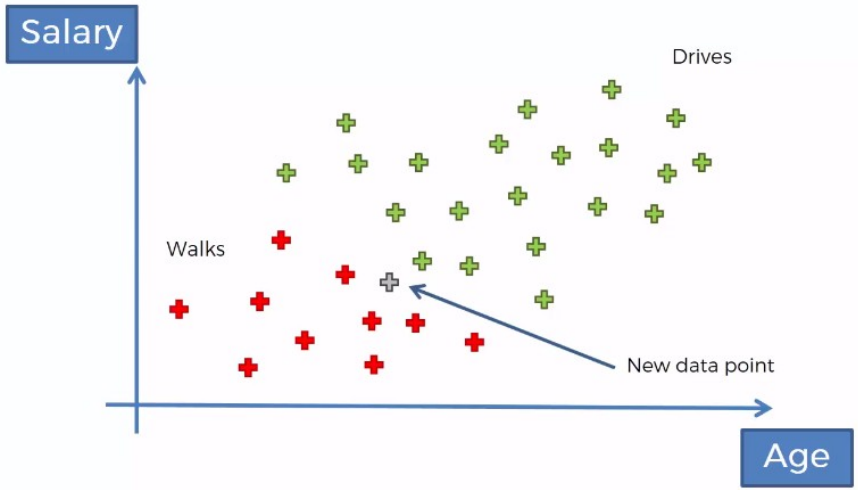
Naive Bayes Classification

$$P(\text{Defect} \mid \text{Mach2}) = \frac{P(\text{Mach2} \mid \text{Defect}) * P(\text{Defect}) * 1000}{P(\text{Mach2}) * 1000} = 1.25\%$$

Let's look at an example:

- 1000 wrenches
- 400 came from Mach2
- 1% have a defect = 10
- of them 50% came from Mach2 = 5
- % defective parts from Mach2 = 5/400 = 1.25%

Naive Bayes Classification



#4 Posterior Probability
#3 Likelihood
#1 Prior Probability

$$P(Walks|X) = \frac{P(X|Walks) * P(Walks)}{P(X)}$$

#2 Marginal Likelihood

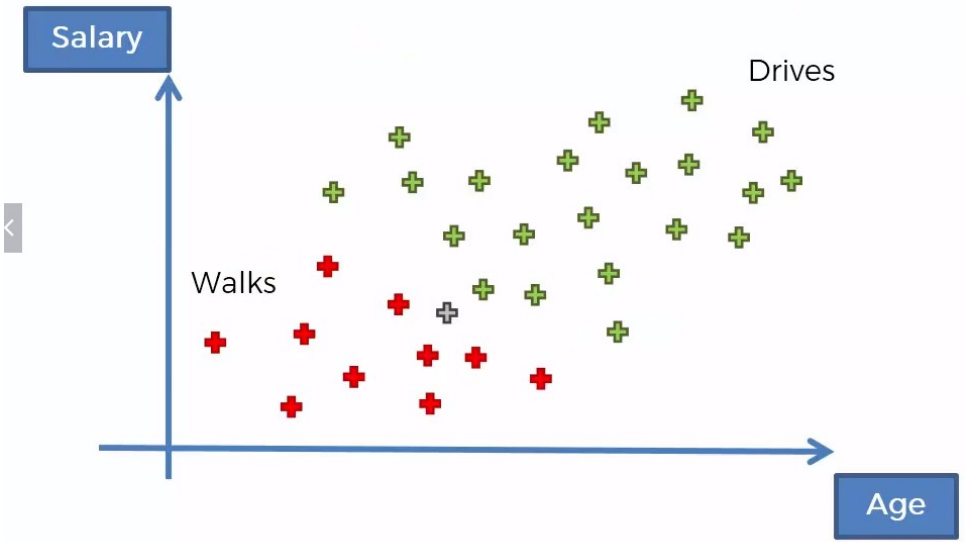
$$P(Walks|X) \; v.s. \; P(Drives|X)$$

#4 Posterior Probability
#3 Likelihood
#1 Prior Probability

$$P(Drives|X) = \frac{P(X|Drives) * P(Drives)}{P(X)}$$

#2 Marginal Likelihood

Naive Bayes Classification



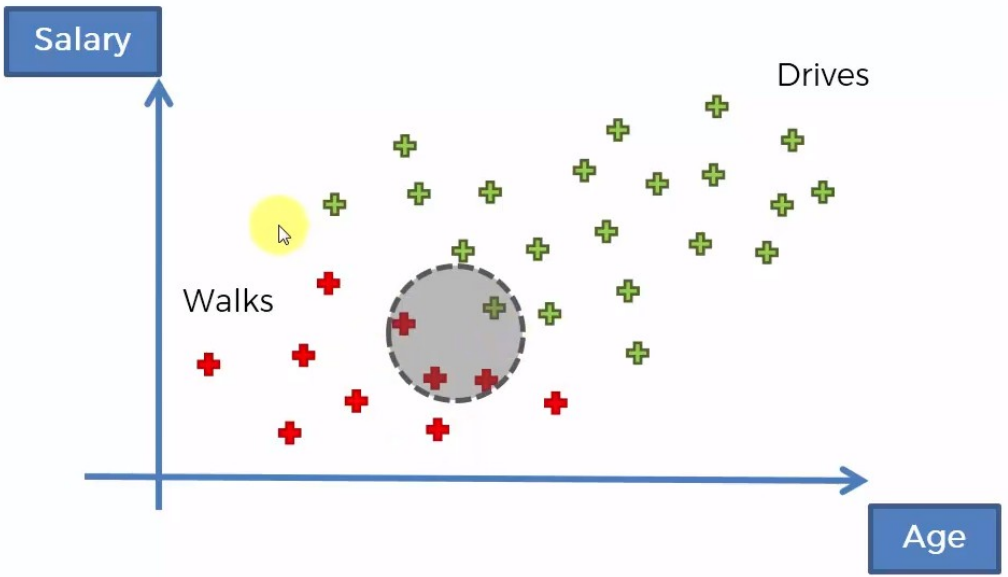
#1. P(Walks)

$$P(Walks) = \frac{Number\ of\ Walkers}{Total\ Observations}$$

$$P(Walks) = \frac{10}{30}$$

Prior Probability

Naive Bayes Classification



#2. P(X)

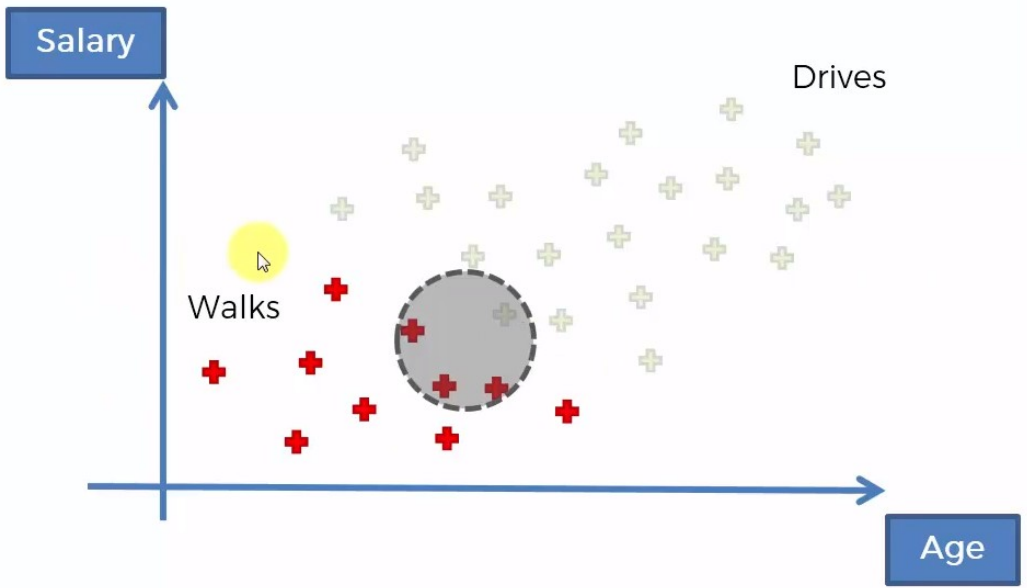
$$P(X) = \frac{Number\ of\ Similar\ Observations}{Total\ Observations}$$

$$P(X) = \frac{4}{30}$$

Imaginary circle with radius you decide but,this radius effect the scores, of course naive bayes accept an optimize radius and make calculations.

Marginal Likelihood

Naive Bayes Classification



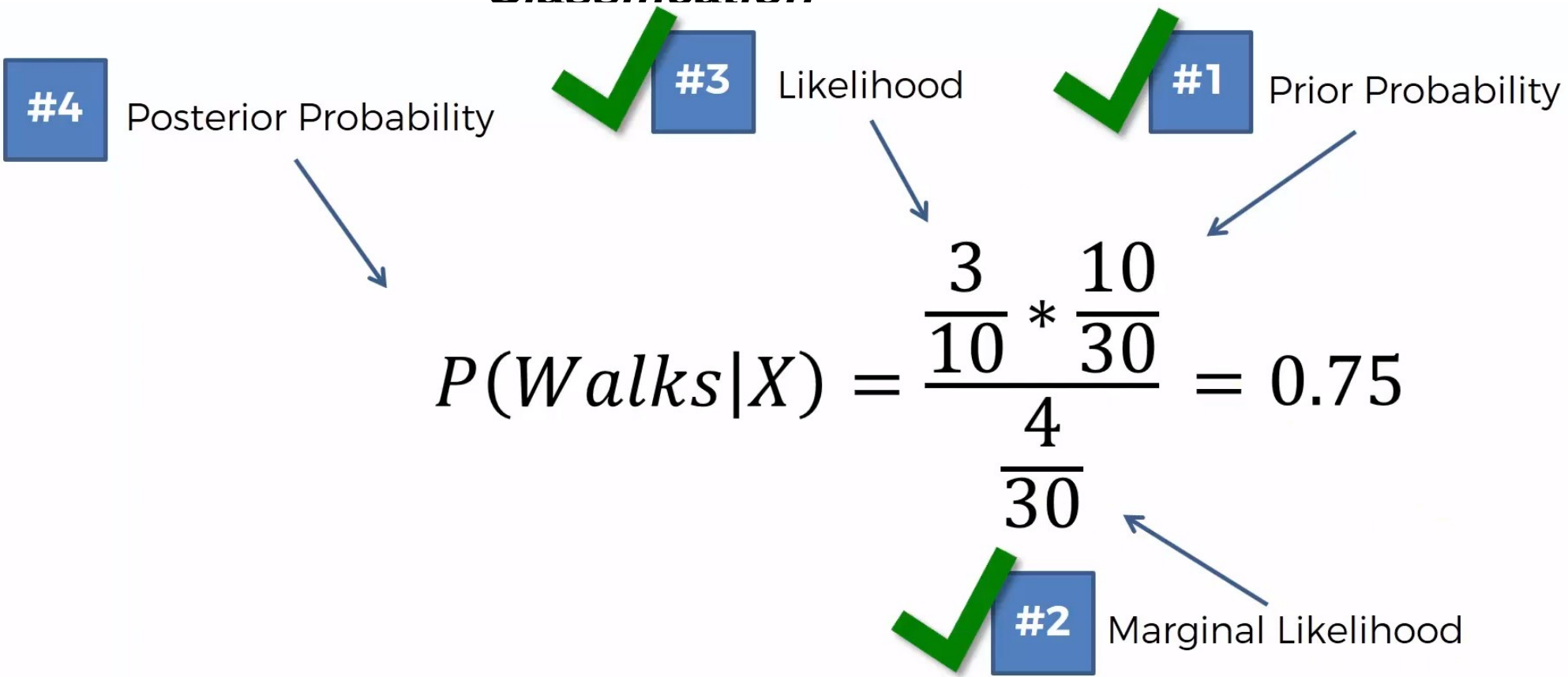
#3. P(X|Walks)

$$P(X|Walks) = \frac{Number\ of\ Similar\ Observations\ Among\ those\ who\ Walk}{Total\ number\ of\ Walkers}$$

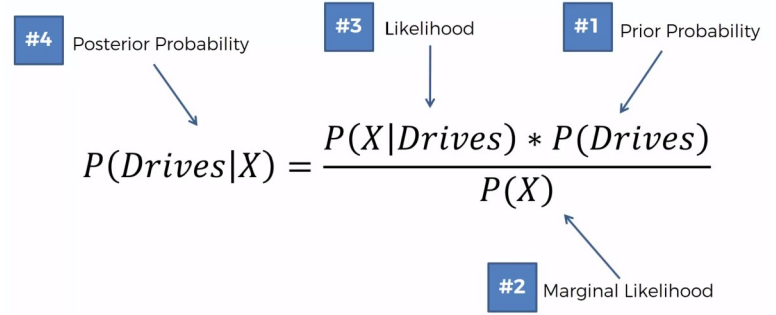
$$P(X|Walks) = \frac{3}{10}$$

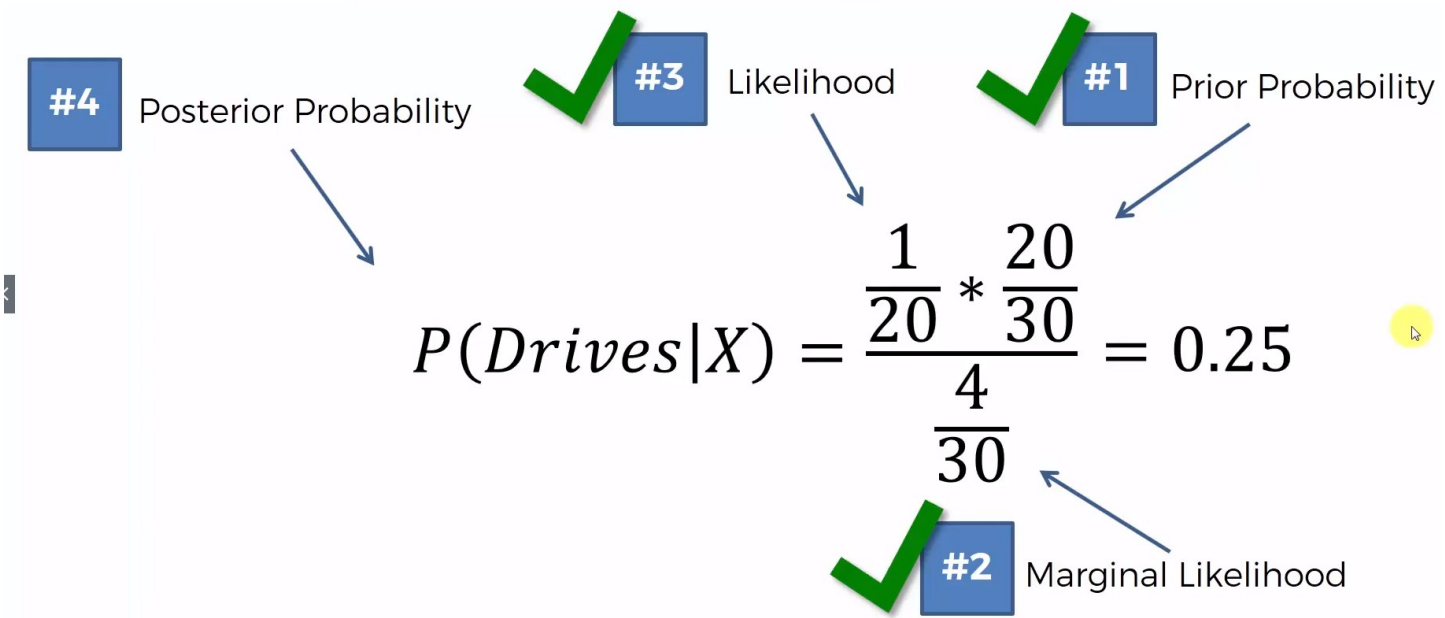
Likelihood

Naive Bayes Classification

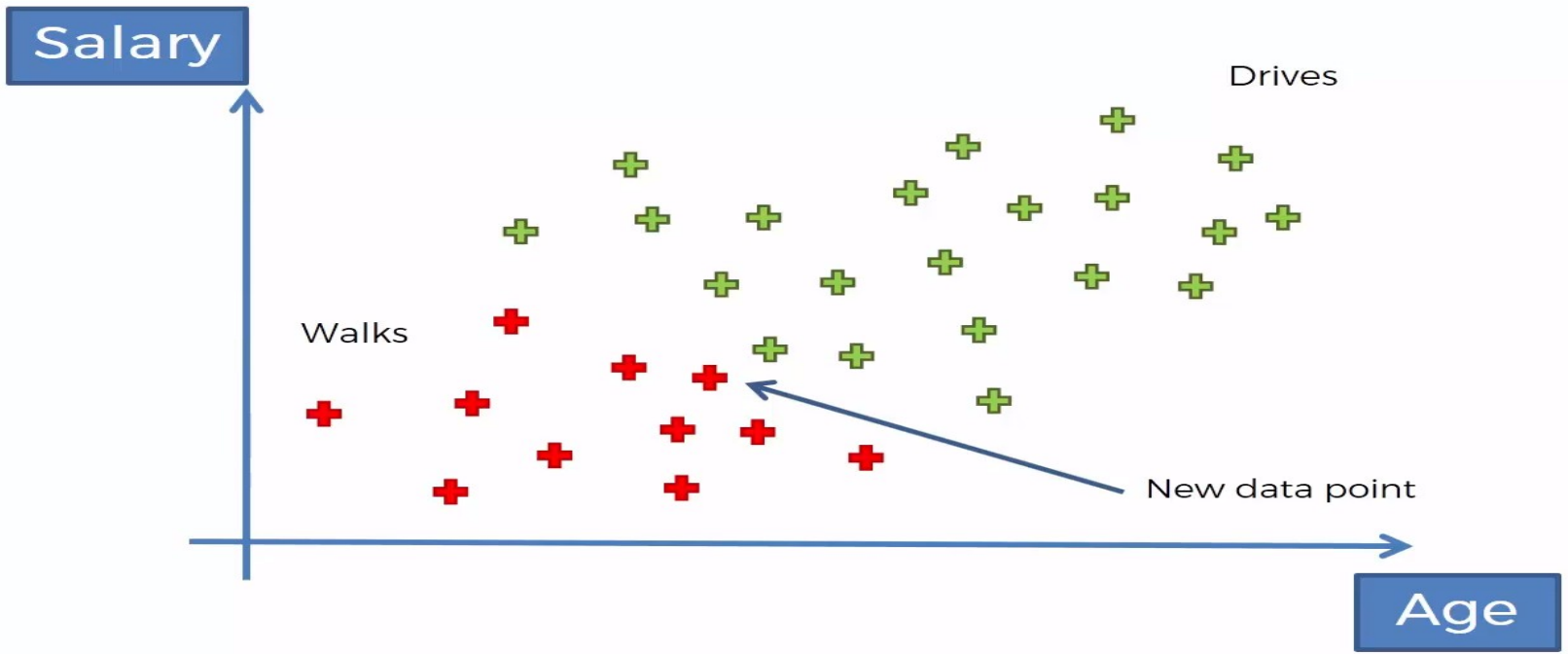


Naive Bayes Classification

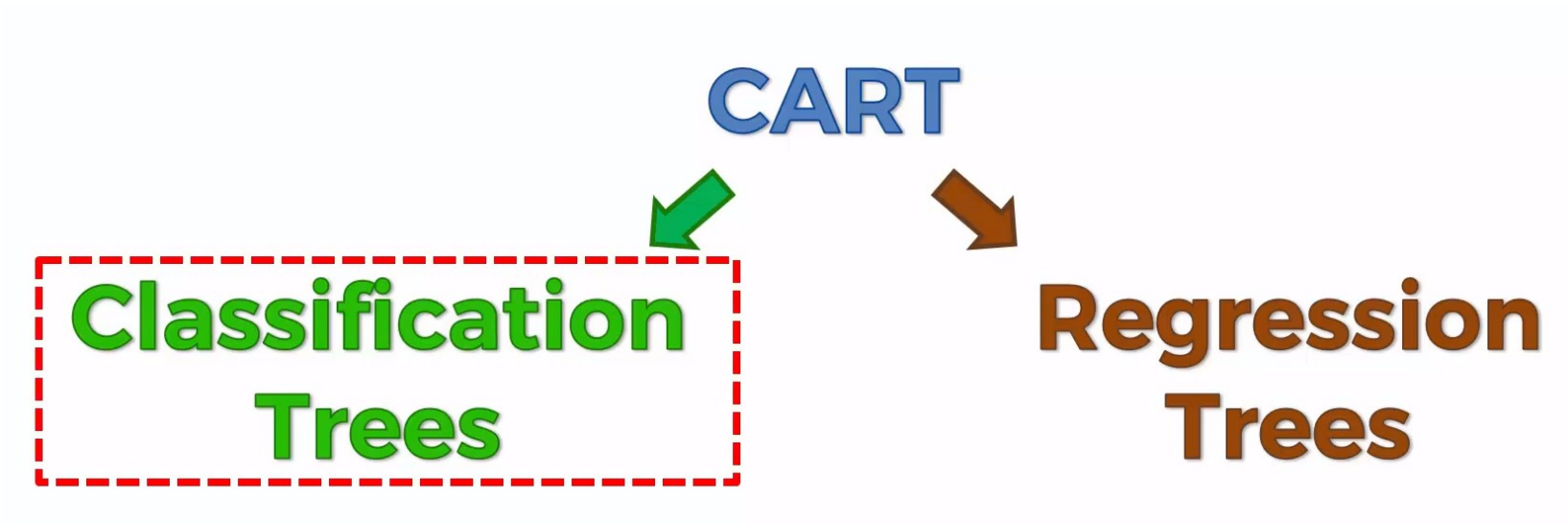




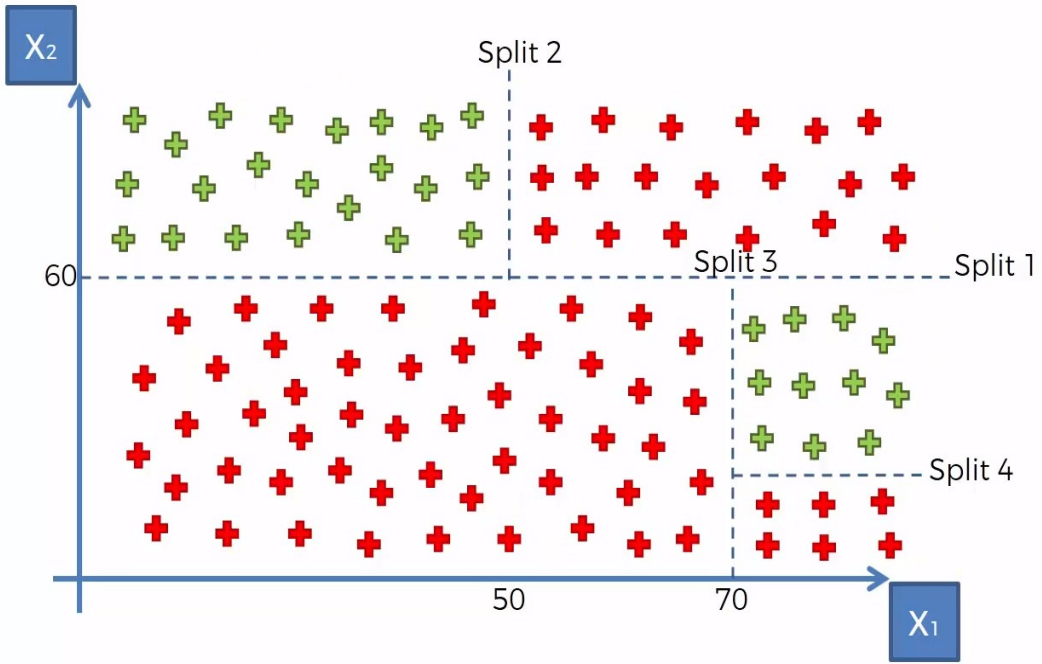
Naive Bayes Classification

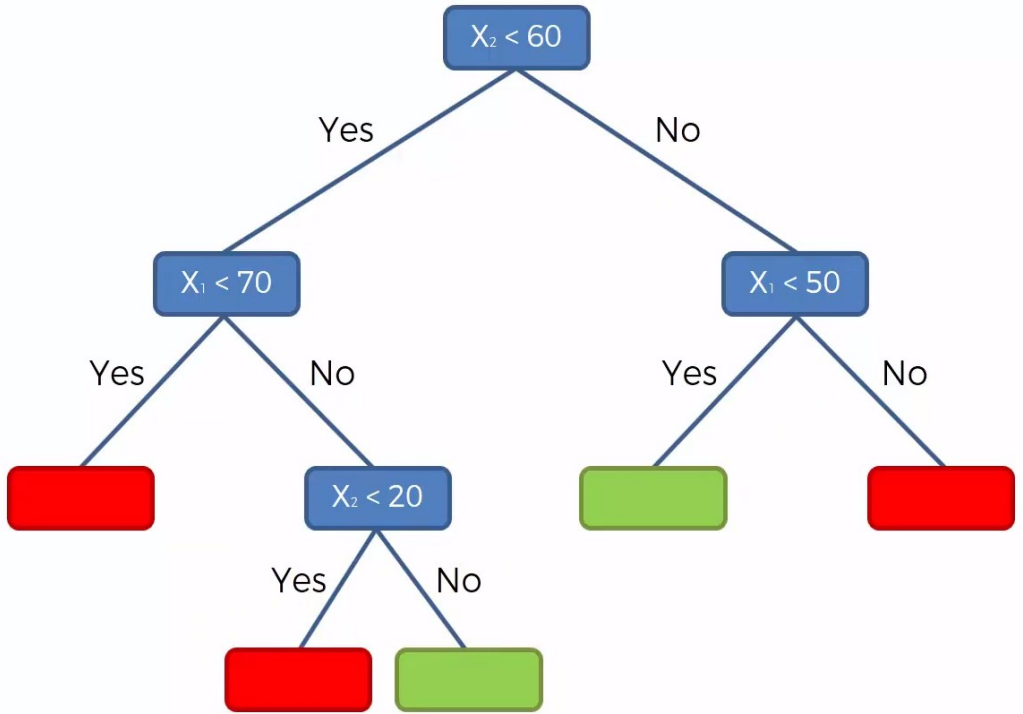


Decision Tree Classification



Decision Tree Classification





Decision Tree Classification

The point is that decision trees are though a very simple tool.

They aren't very powerful on their own but they're used in other methods that leverage their simplicity and create some very powerful machine learning algorithms and such algorithms even are used to perform facial recognition like on your iPhone.

It is quite a simple method but at the same time it lies in the foundation of some of the more modern and more powerful methods in machine learning.

Random Forest Classification

STEP 1: Pick at random K data points from the Training set.

STEP 2: Build the Decision Tree associated to these K data points.

STEP 3: Choose the number Ntree of trees you want to build and repeat STEPS 1 & 2

STEP 4: For a new data point, make each one of your Ntree trees predict the category to which the data points belongs, and assign the new data point to the category that wins the majority vote.

Random Forest Classification

Random Forest Classification

No Joystick
No Steering wheel

Random Forest Classification

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. *Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.*

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

1. Introduction

Robust interactive human body tracking has applications including gaming, human-computer interaction, security, telepresence, and even health-care. The task has recently been greatly simplified by the introduction of real-time depth cameras [16, 19, 44, 37, 28, 13]. However, even

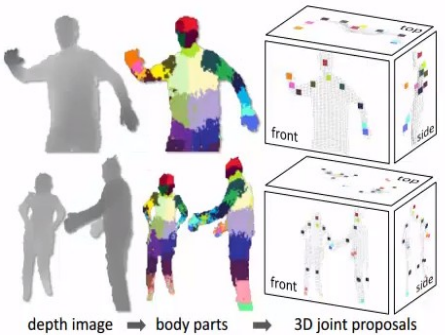


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

joints of interest. Reprojecting the inferred parts into world space, we localize spatial modes of each part distribution and thus generate (possibly several) confidence-weighted proposals for the 3D locations of each skeletal joint.

We treat the segmentation into body parts as a per-pixel classification task (no pairwise terms or CRF have proved necessary). Evaluating each pixel separately avoids a com-

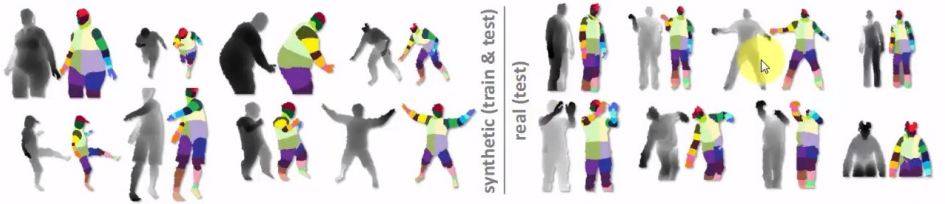


Figure 2. **Synthetic and real data**. Pairs of depth image and ground truth body parts. Note wide variety in pose, shape, clothing, and crop.

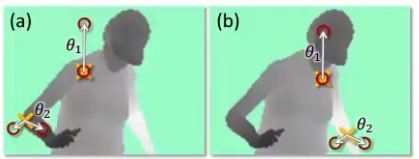
simplify the task of background subtraction which we assume in this work. But most importantly for our approach, it is straightforward to synthesize realistic depth images of people and thus build a large training dataset cheaply.

2.2. Motion capture data

The human body is capable of an enormous range of poses which are difficult to simulate. Instead, we capture a large database of motion capture (mocap) of human actions

the appearance variations we hope to recognize at test time. While depth/scale and translation variations are handled explicitly in our features (see below), other invariances cannot be encoded efficiently. Instead we learn invariance from the data to camera pose, body pose, and body size and shape.

The synthesis pipeline first randomly samples a set of parameters, and then uses standard computer graphics techniques to render depth and (see below) body part images



igure 3. **Depth image features.** The yellow crosses indicates the xel $\mathbf{x}$ being classified. The red circles indicate the offset pixels defined in Eq. 1. In (a), the two example features give a large epth difference response. In (b), the same two features at new nage locations give a much smaller response.

arts for left and right allow the classifier to disambiguate ie left and right sides of the body.

Of course, the precise definition of these parts could be anged to suit a particular application. For example, in an pper body tracking scenario, all the lower body parts could

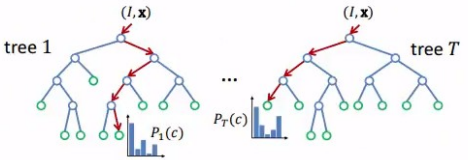
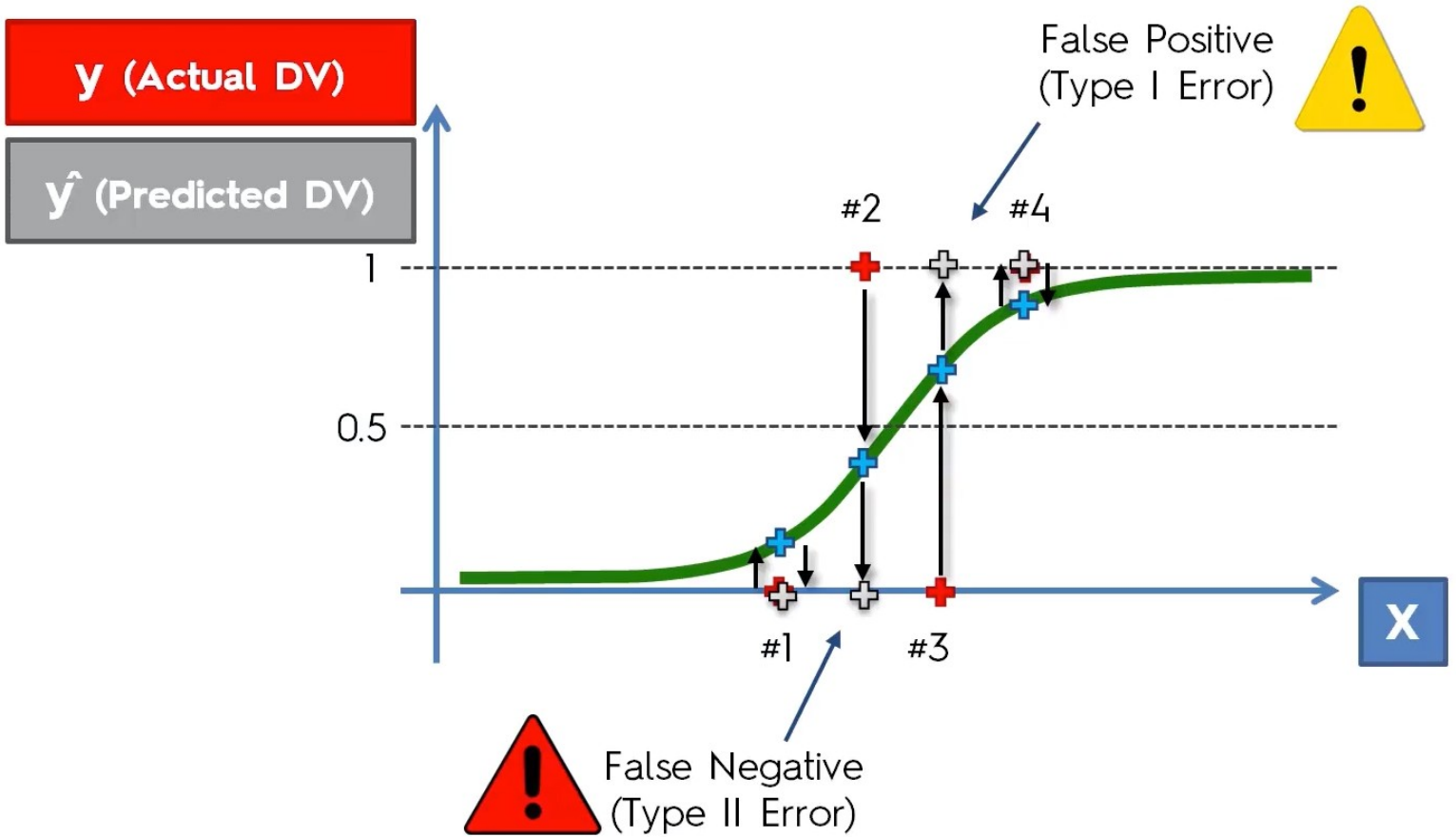


Figure 4. **Randomized Decision Forests.** A forest is an ensemble of trees. Each tree consists of split nodes (blue) and leaf nodes (green). The red arrows indicate the different paths that might be taken by different trees for a particular input.

3.3. Randomized decision forests

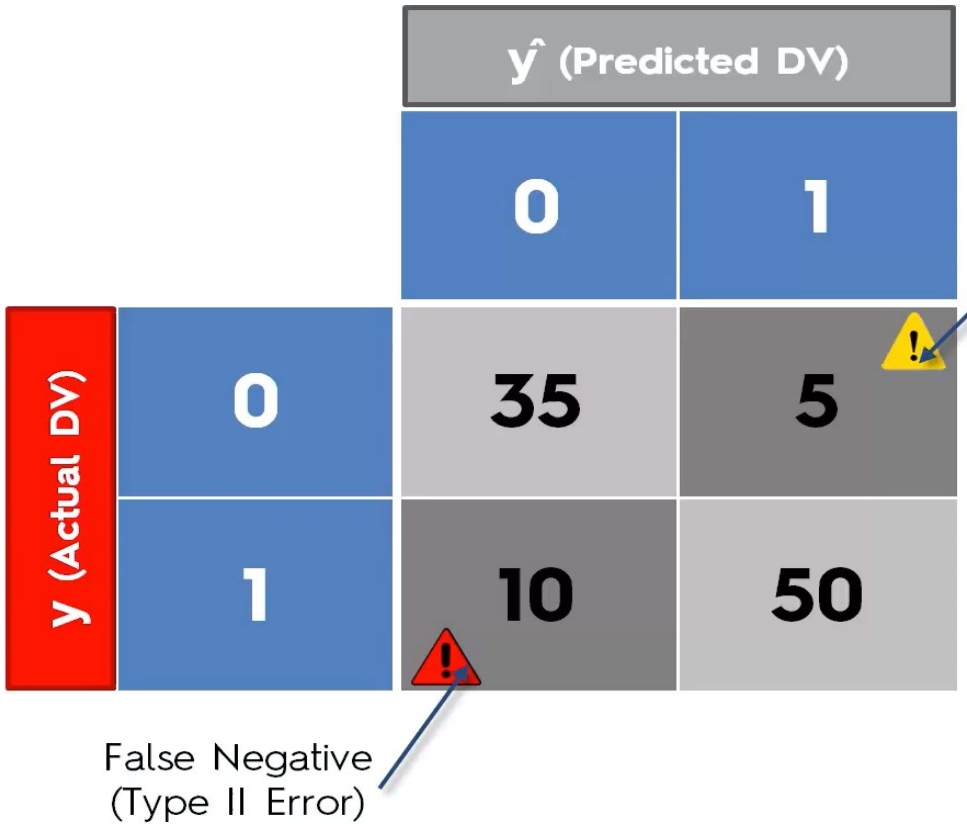
Randomized decision trees and forests [35, 30, 2, 8] have proven fast and effective multi-class classifiers for many tasks [20, 23, 36], and can be implemented efficiently on the GPU [34]. As illustrated in Fig. 4, a forest is an ensemble of T decision trees, each consisting of split and leaf nodes.

Classification Models Performance Evaluation



Classification Models Performance Evaluation

Confusion Matrix



Calculate two rates

1. Accuracy Rate = Correct / Total
AR = 85/100 = 85%

2. Error Rate = Wrong / Total
ER = 15/100 = 15%

Classification Models Performance Evaluation

		$\hat{y}$ (Predicted DV)	
		0	1
y (Actual DV)	0	9,700	150 ⚠
	1	50 ⚠	100

Scenario 1:

Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%

Classification Models Performance Evaluation

y (Actual DV)	ŷ (Predicted DV) = 0	ŷ (Predicted DV) = 1
0	9,700 ←	150 !
1	50 ! ←	100

You should not base your judgment just on accuracy right, because things like this can happen and even though obviously you're not using a model any more which means that you're not applying any kind of logic into your decision making process; your accuracy rate is going up, so it's misleading you into a wrong conclusion that you should stop using models.

This effect is called the *accuracy paradox*.

y (Actual DV)	ŷ (Predicted DV) = 0	ŷ (Predicted DV) = 1
0	9,850 ←	0 !
1	150 ! ←	0

Scenario 1:

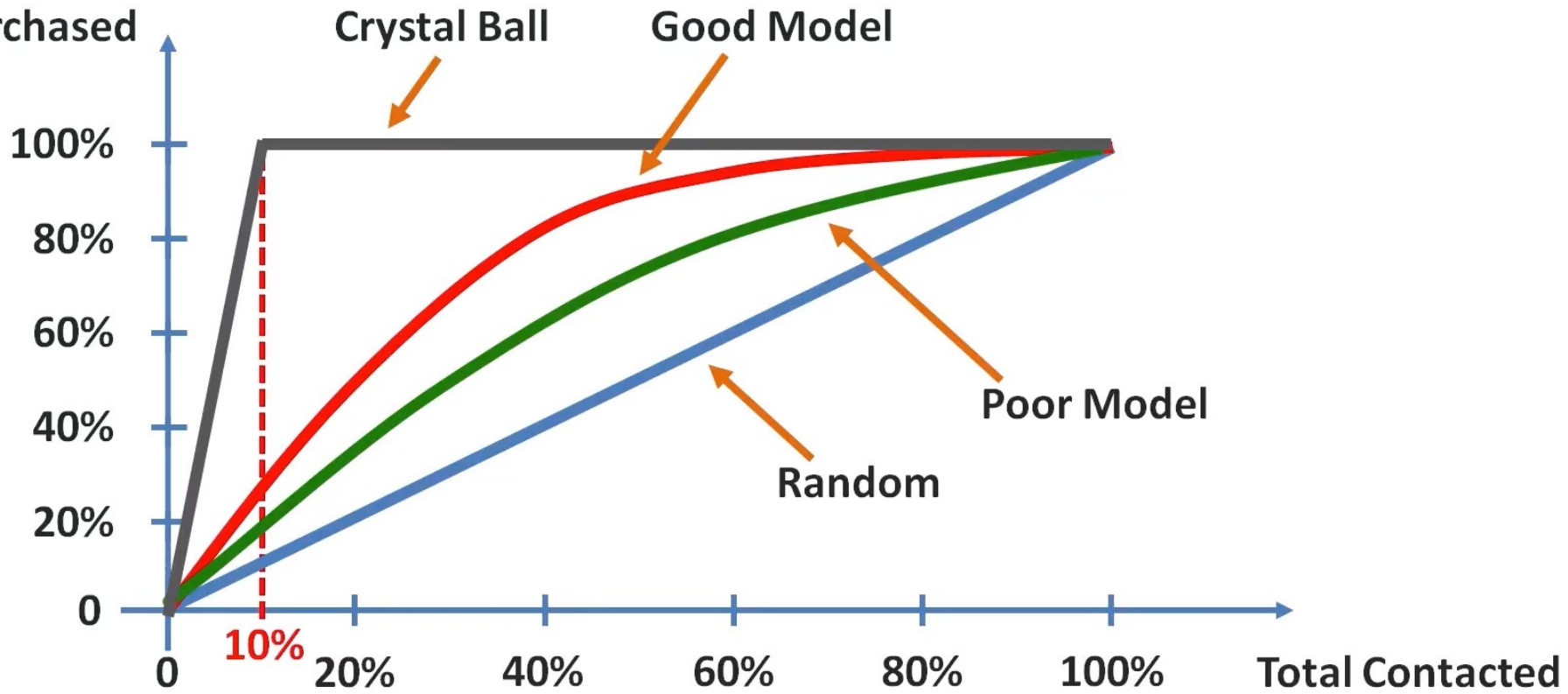
Accuracy Rate = Correct / Total
AR = 9,800/10,000 = 98%

Scenario 2:

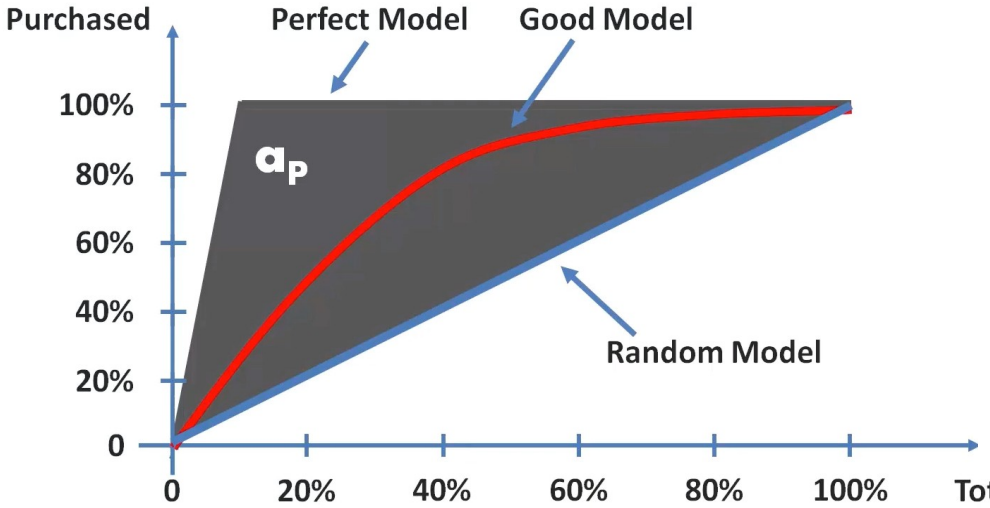
Accuracy Rate = Correct / Total
AR = 9,850/10,000 = 98.5% ↑

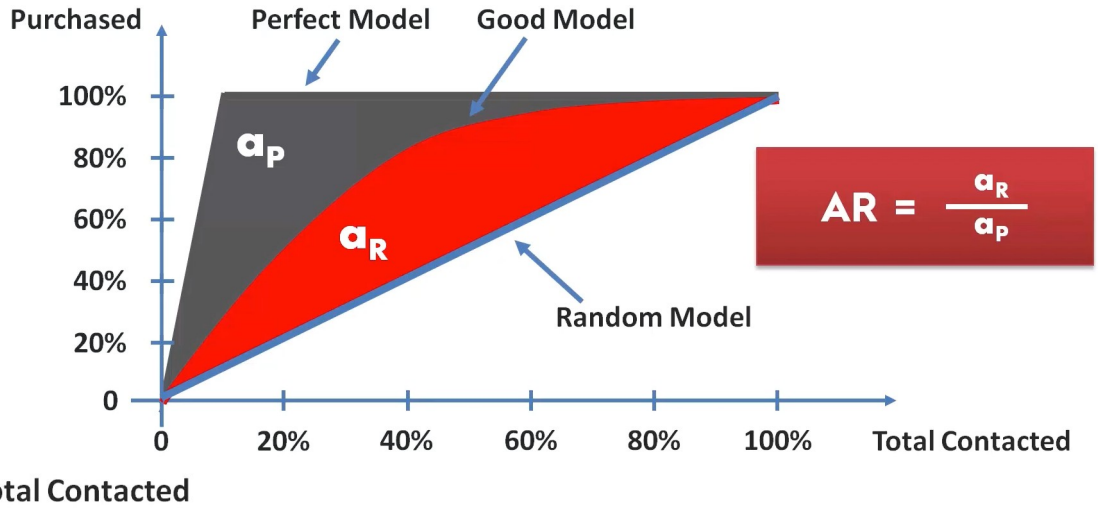
Classification Models Performance Evaluation

CAP Curve(Cumulative Accuracy P



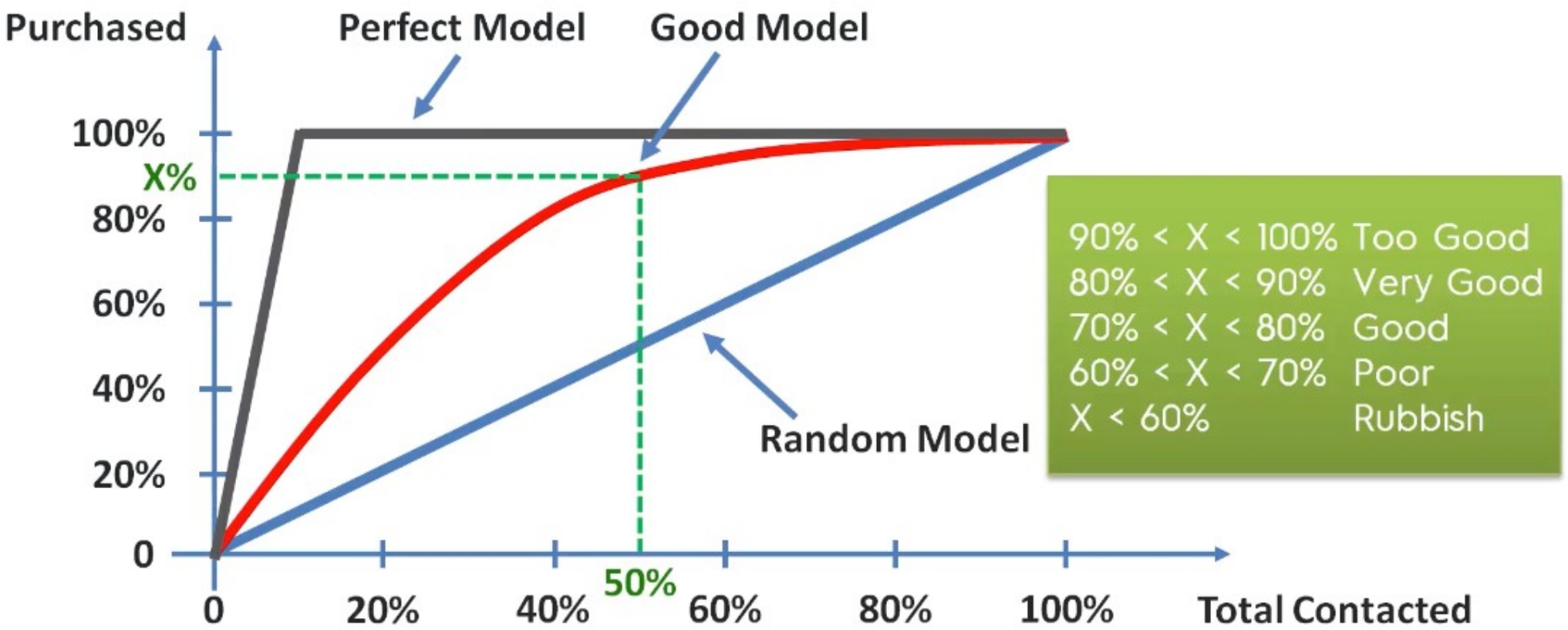
Classification Models Performance Evaluation





0 < AR < 1
AR ~ 0 Very Bad Model
AR ~ 1 Very Good Model
AR = 1 Overfitting

Classification Models Performance Evaluation



Classification Models Performance Evaluation

How do I know which model to choose for my problem ?

You first need to figure out whether your problem is linear or non linear.

If your problem is linear, you should go for Logistic Regression or SVM.7

If your problem is non linear, you should go for K-NN, Naive Bayes, Decision Tree or Random Forest.

Logistic Regression or Naive Bayes when you want to rank your predictions by their probability. For example if you want to rank your customers from the highest probability that they buy a certain product, to the lowest probability. Eventually that allows you to target your marketing campaigns. And of course for this type of business problem, you should use *Logistic Regression* if your problem is linear, and *Naive Bayes* if your problem is non linear.

***SVM* when you want to predict to which segment your customers belong to. Segments can be any kind of segments, for example some market segments you identified earlier with clustering.**

***Decision Tree* when you want to have clear interpretation of your model results.**

***Random Forest* when you are just looking for high performance with less need for interpretation.**

See You Next Week :)

Next Week Topic :

Clustering

Python Codes (Regression / Classification)