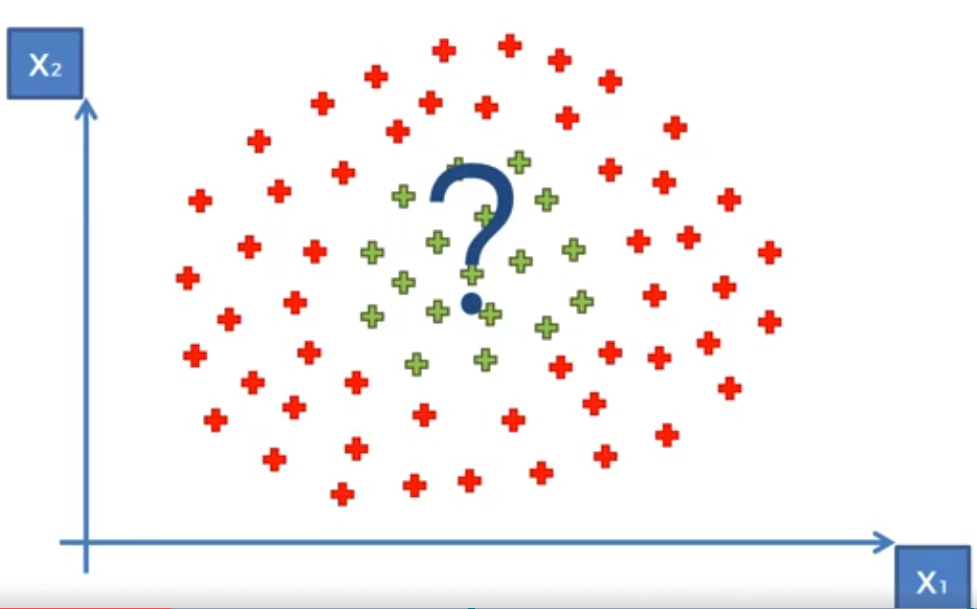
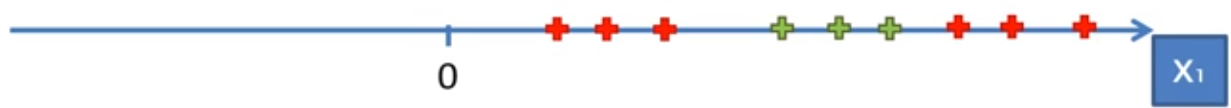
***Kernel SVM***

* What happens if data isn’t linearly separable?



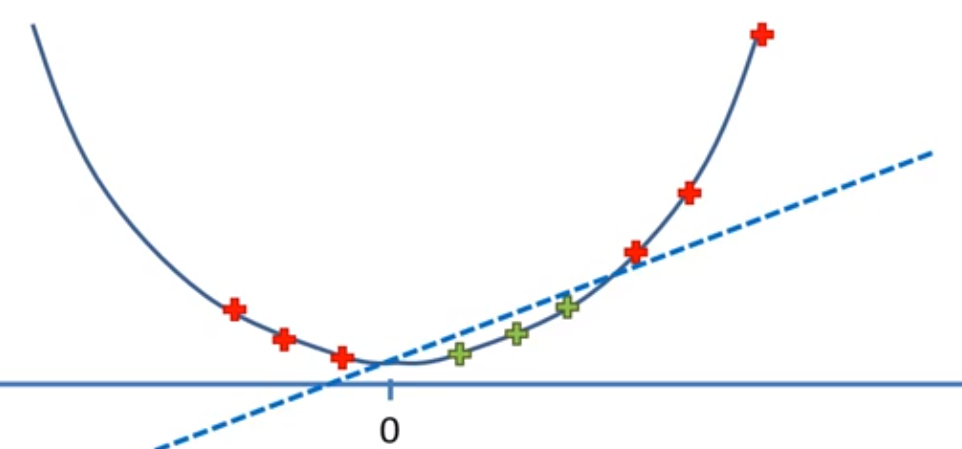
* 1D space, non-linearly separable dataset:



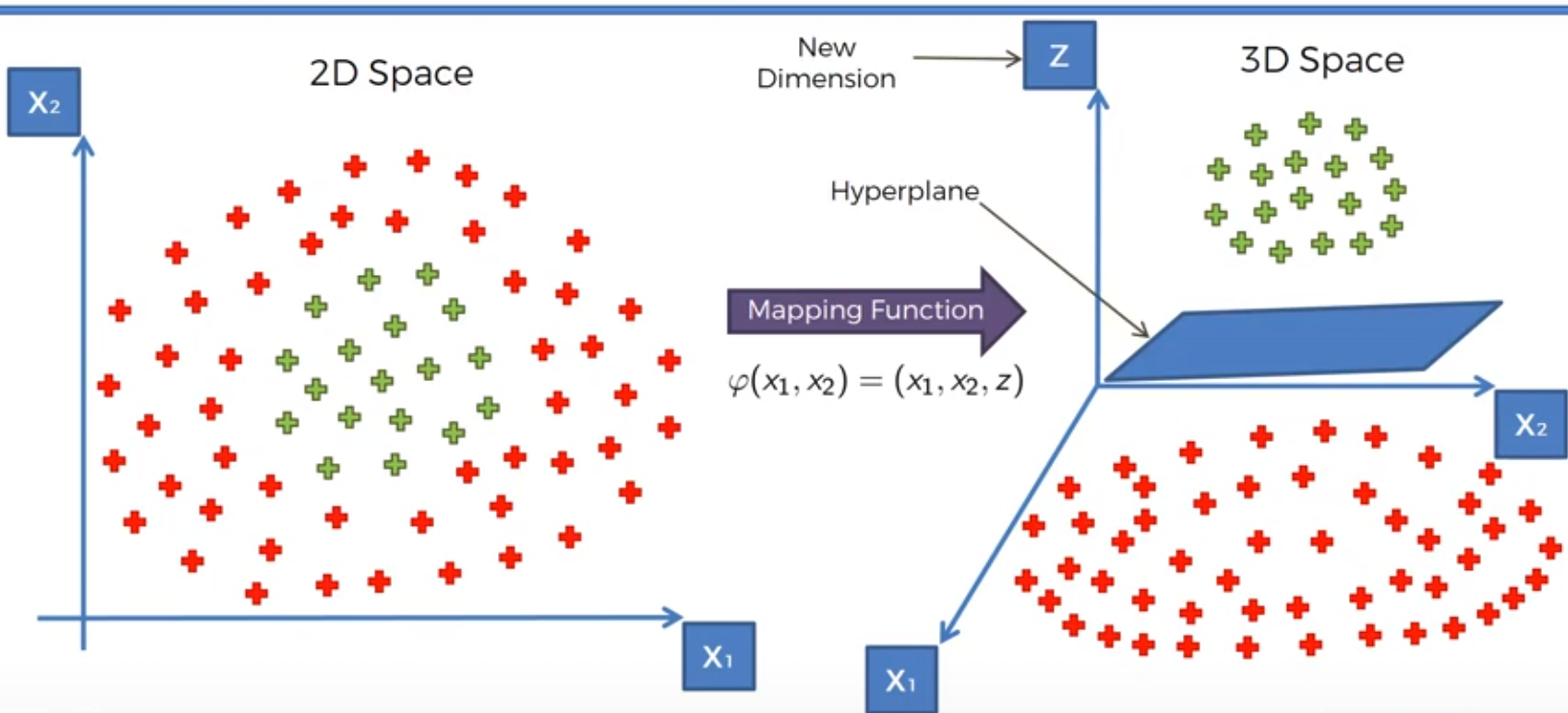
* Want to **increase dimensionality** of the space to make the dataset linearly separable in a higher-dimensional space
* 1st step: Start to build a **mapping function** 🡪 say, f = x – 5
* This move all DP’s to the left 5 spots



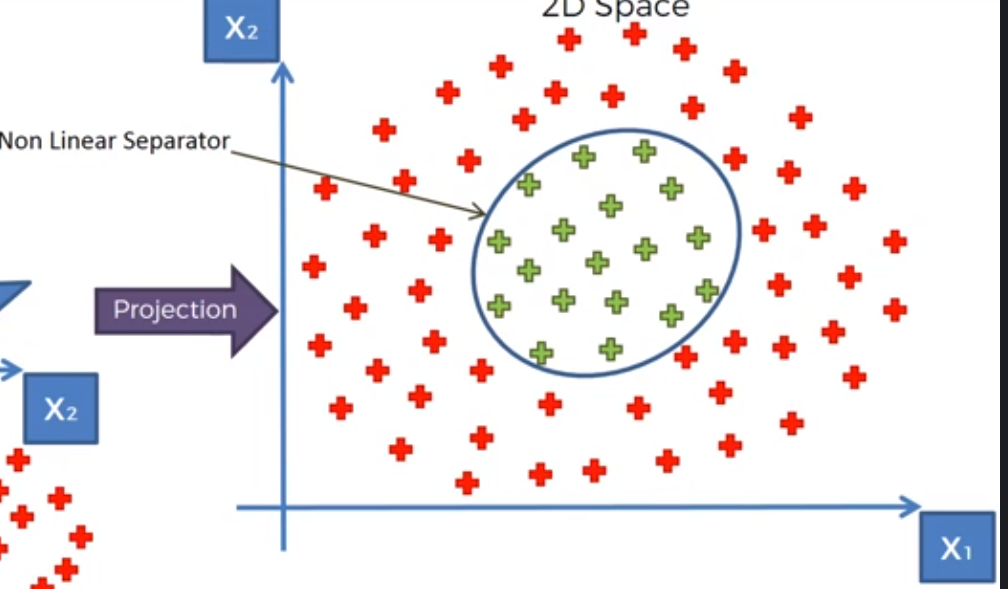
* Then square it all f = (x-5)^2 = 2 dimensional 🡪 see that data is now linearly separable:



* Then we project this back into 1D space
* Apply same principle to a 2D space to somehow map it to a 3D space:
* Then see that the data is linearly separable by a **hyperplane**



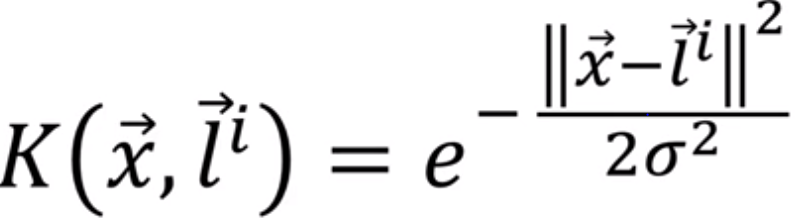
* Then we project this hyperplane back into a 2D space to see our non-linear classifier



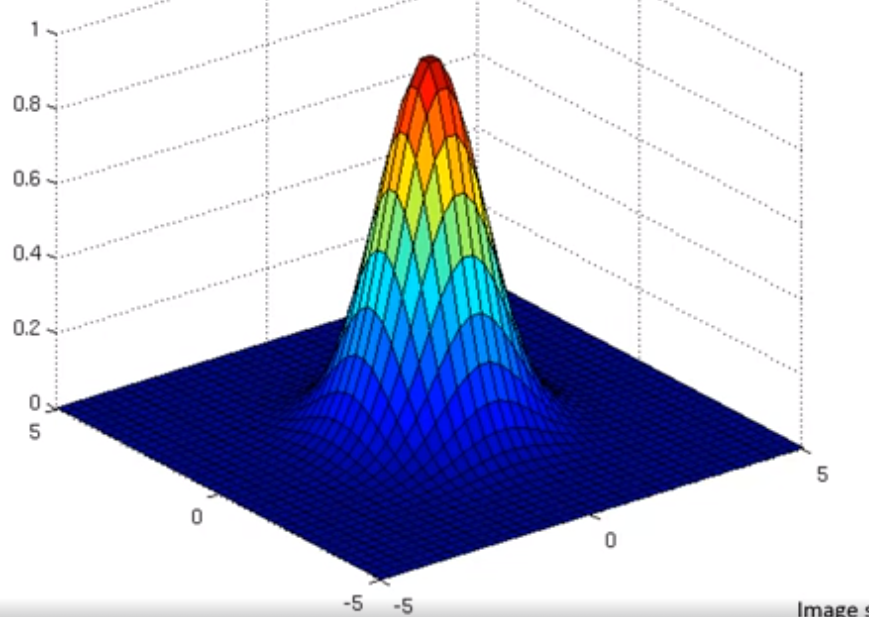
* Possible issue 🡪 mapping to higher-dimensional spaces = computationally expensive, especially w/ higher dataset
* Therefore, this approach isn’t the most efficient
* Going to use the **kernel trick** to deal w/ this to get similar results w/out mapping to higher dimensions

**Kernel Trick**

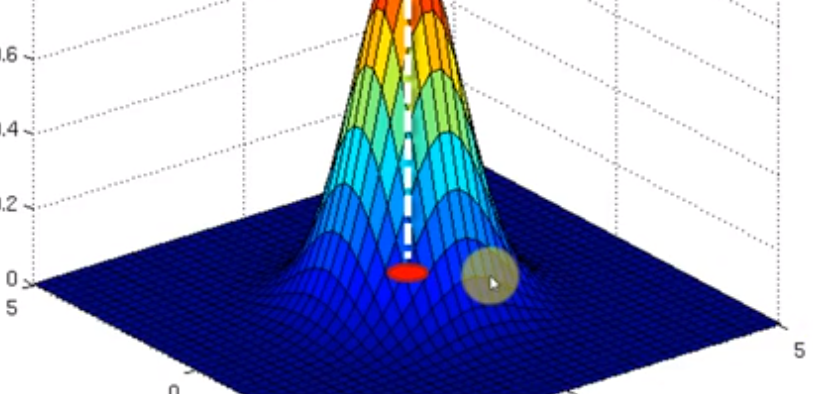
* **(Gaussian) Radial Basis Function**



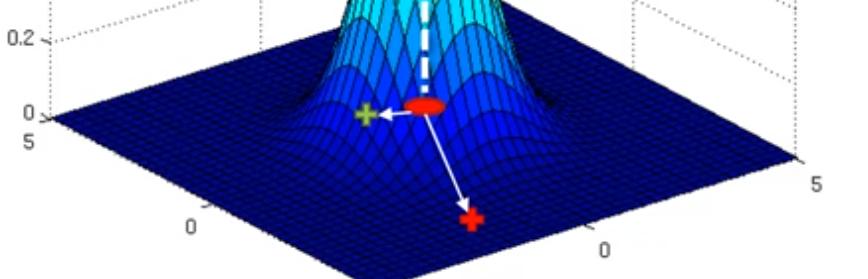
* K “kernel” = a function of 2 vectors = x (some point in the data), l^i (**landmark i**)
* These are **feature vectors** in some input space,
* Then, e is raised to the squared Euclidean distance (greater weight on objects that are farther apart) over 2 times the **free parameter**, sigma, squared
* Here’s the function for a specific landmark (in the middle of the 2D space, {0,0}) + specific sigma



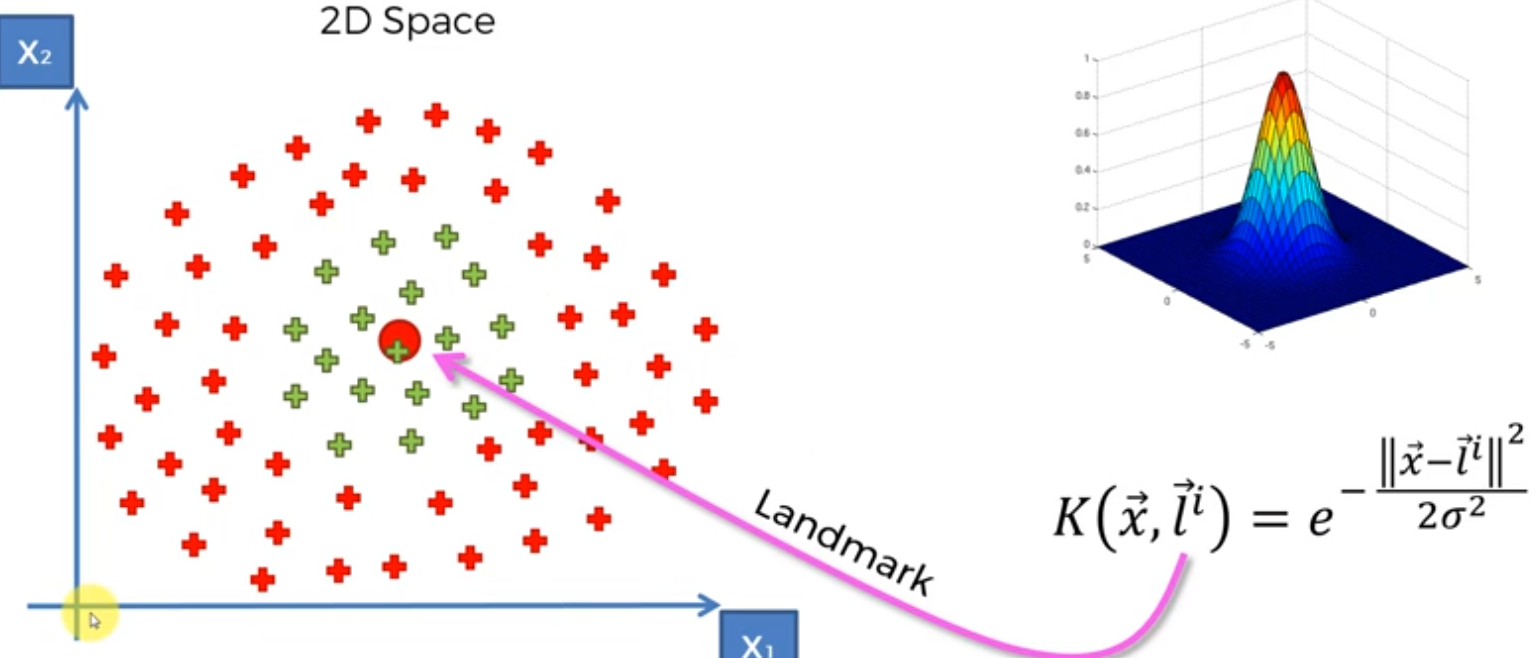
* Vertical height = result of the **Gaussian RBF** for every other point on the 2D plane (x)
* So, we find the distance between our landmark l\_i + those points, x, + then perform the function
* If we take the tip of this shape in the middle of the 2D plane + project it onto the plane, we get the landmark



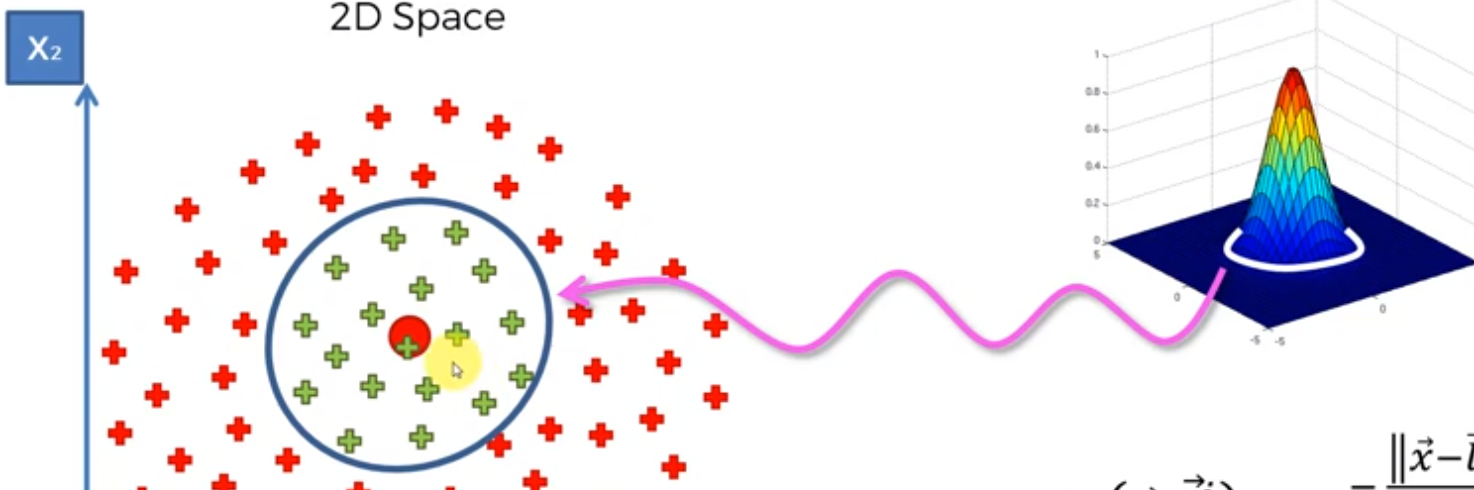
* This is from where K is measured (i.e. from this landmark to a point, x)



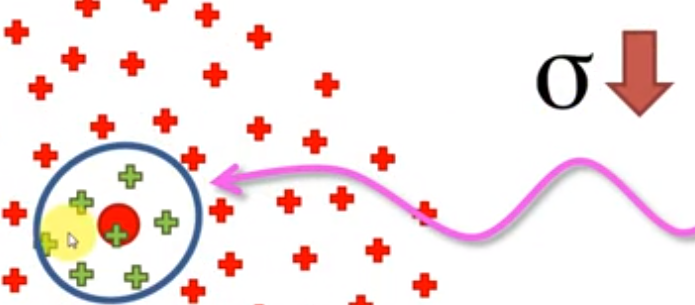
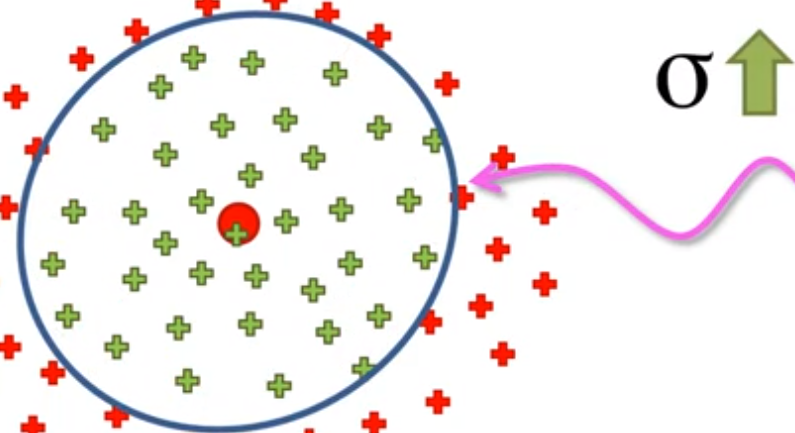
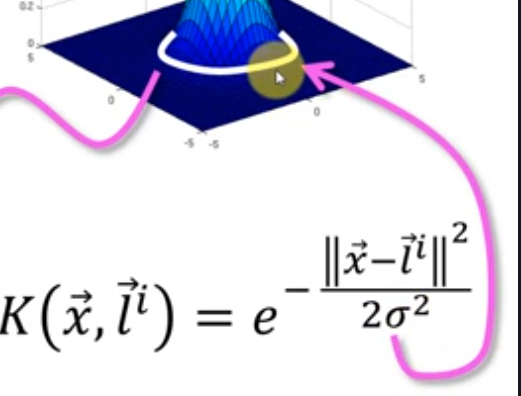
* The red x distance looks large, so the result is large + the square is large, divided by sigma, it can still be large (depending on sigma), + then taking e to the very large, negative exponent, we end up w/ 1 over e raised to a very large number, which results in a very small number
* So, x = far away from landmark, K/height = very close to 0
* Vice versa for small distances (blue x) 🡪 as x gets closer to l\_i, the power e is raised to get closer to 0, so K/heights gets closer to 1
* We will use this kernel function to separate/classify our DP’s
* Take the landmark + put it somewhere in our 2D space of data (a whole methodology of how the algorithm does this in R or Python = way to find optimal place for landmarks)



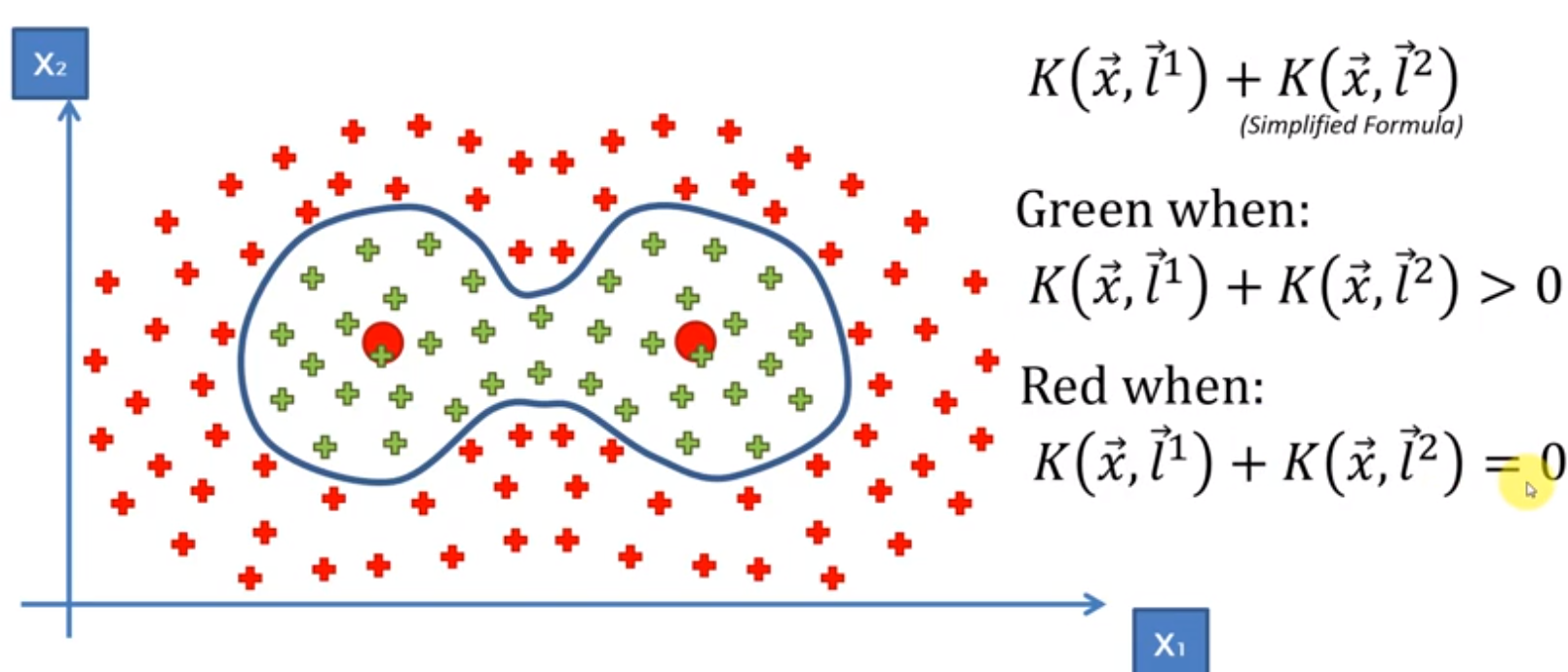
* Then, the circle around the base of the kernel function shape is projected



* This circumference allows us to find all points w/in the circumference + give them a value > 0 + everything outside it will be = 0 or very close to 0
* **Sigma is what defines the circumference**



* By finding the *right* sigma, we can set up the correct kernel function to classify our DP’s
* So, we’ve created a decision boundary w/out going to a higher dimensional space or creating a mapping function, etc.
* We have a visual representation that is in a higher-dimensional space, but the actual computations are being done in the lower-dimensional space
* This can help us solve even more complex problems by doing things like “adding together” 2 kernel functions



* AS we move away from landmark 1, the kernel function gets closer to 0, and same for kernel 2, so they don’t really interfere here
* *VERY SIMPLIFIED EXAMPLE*

**Types of Kernel Functions**

* 
* 
* Still select a landmark + different distances fmro this landmark cause different results
* Can see that anything to the right = higher value and vice versa to the left
* 
* Can view kernel types in 3D at **htps://mlkernels.readthedocs.io/en/latest/kernelfunctions.html**