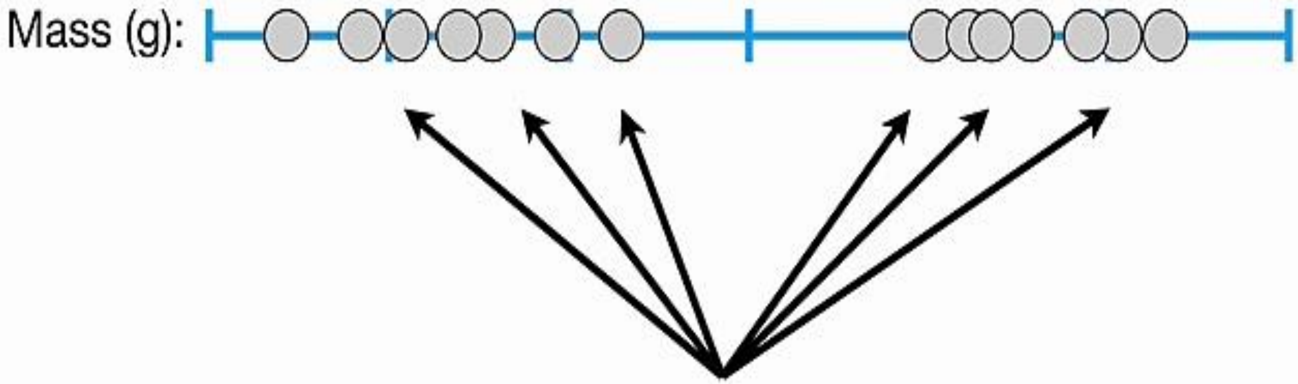
DATA MINING 2
Support Vector Machine

Riccardo Guidotti

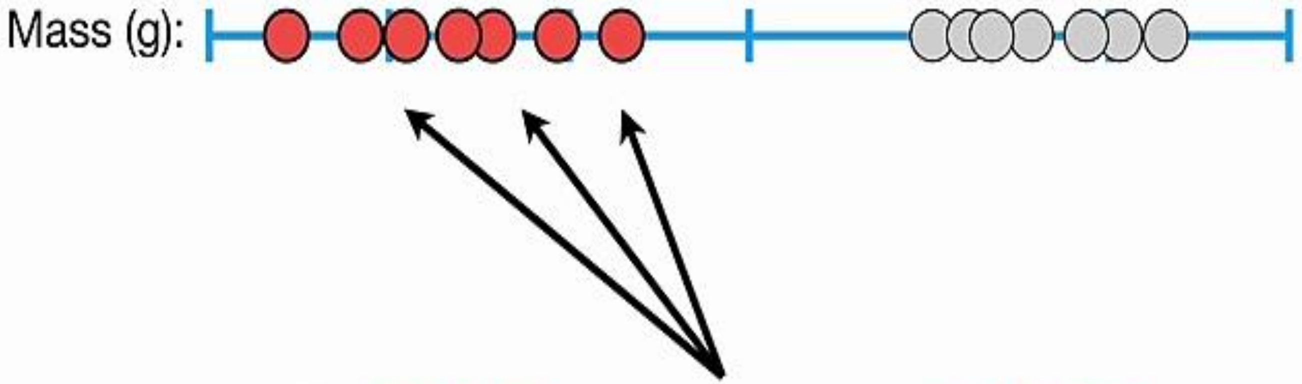
a.a. 2023/2024

Slides edited from Tan, Steinbach, Kumar, Introduction to Data Mining
Contains slides integrated with StatQuest

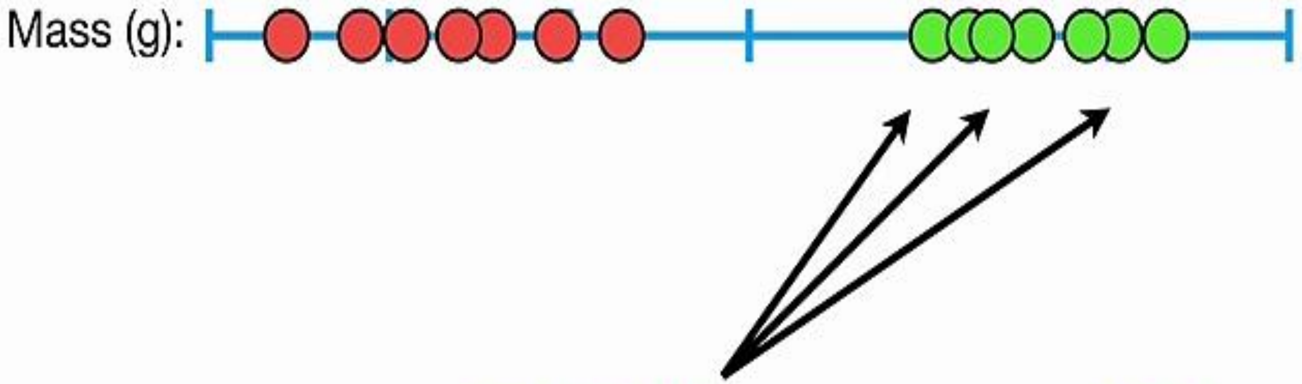
UNIVERSITÀ DI PISA



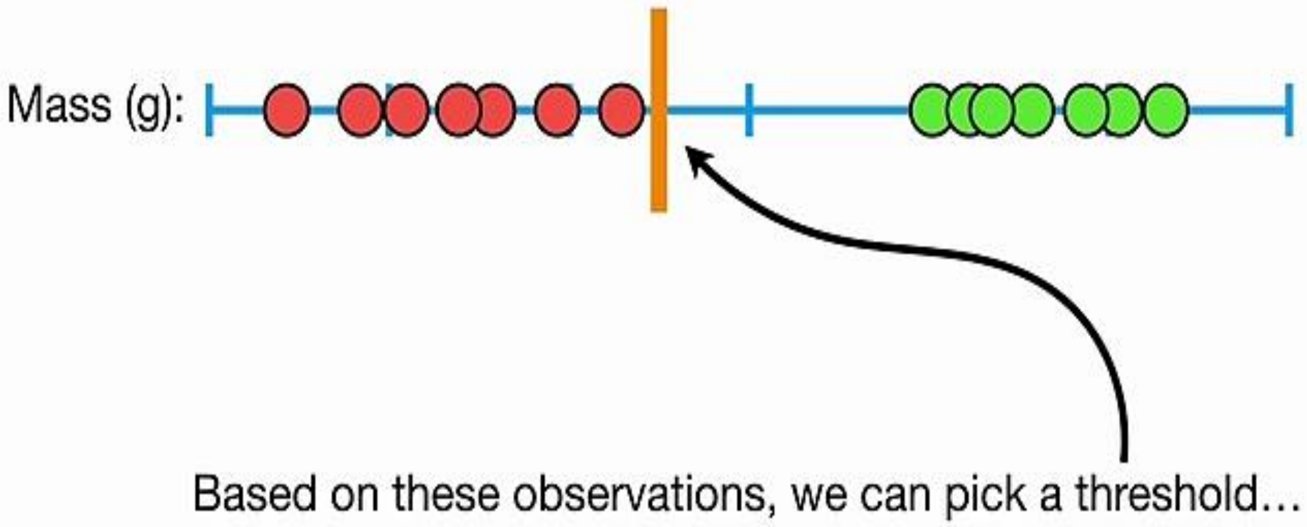
Let's start by imagining we measured
the mass of a bunch of mice…

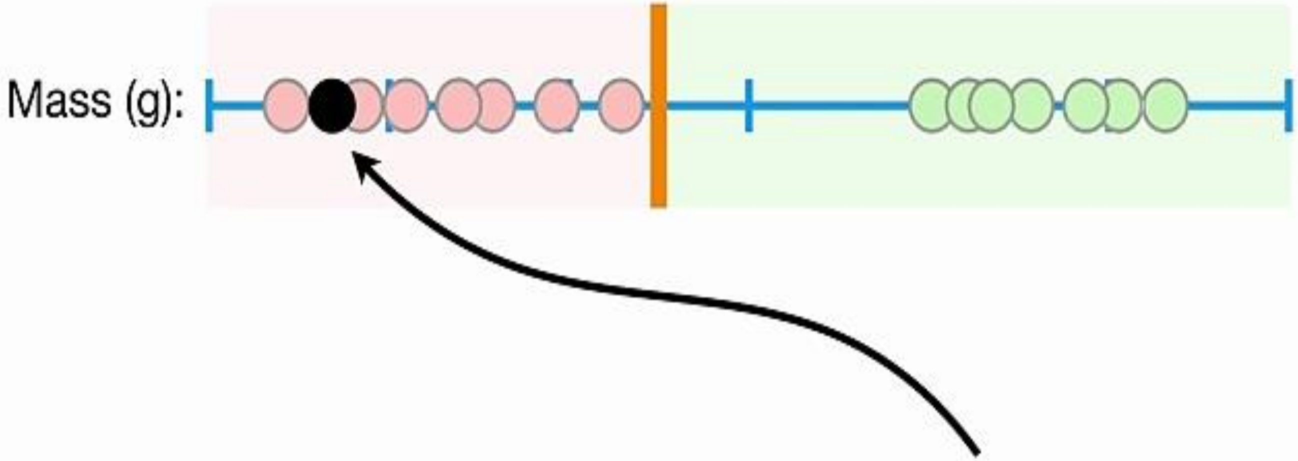


The **red dots** represent mice are ***not obese***…

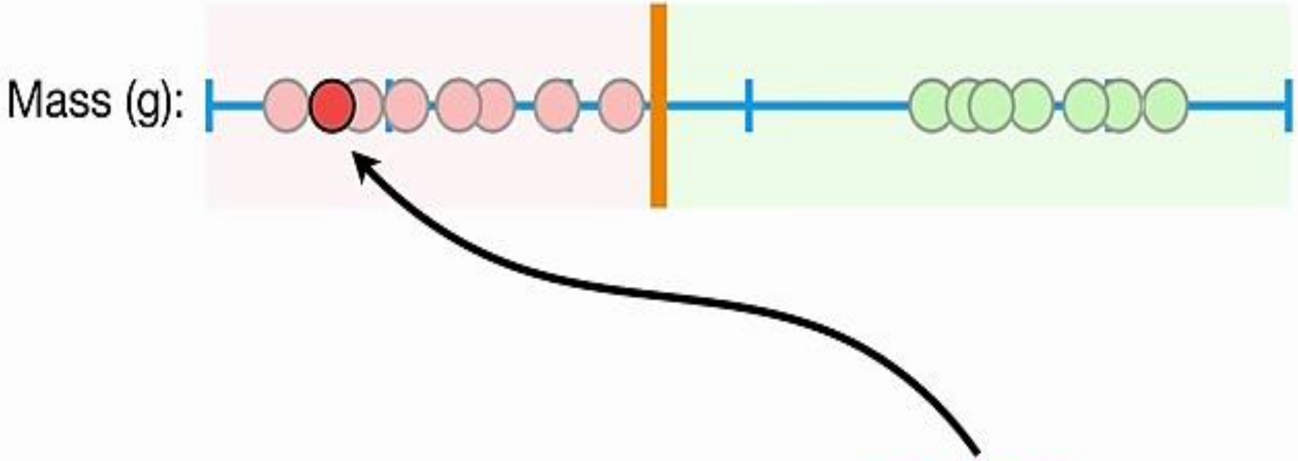


...and the **green dots** represent mice are ***obese***.

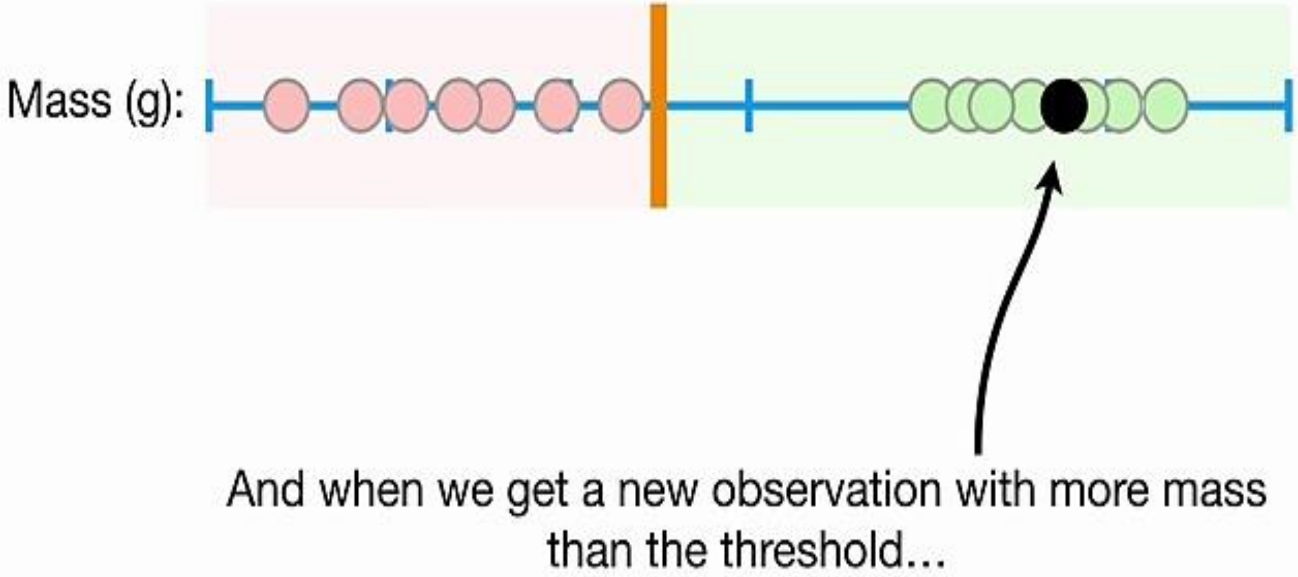

Mass (g):
Based on these observations, we can pick a threshold…

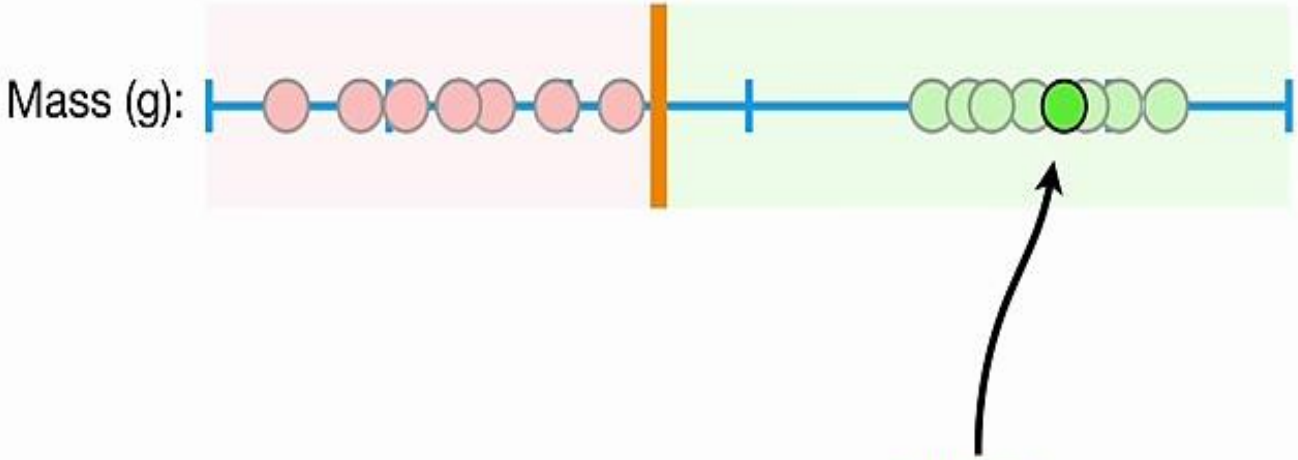


...and when we get a new observation that has less mass than the threshold...

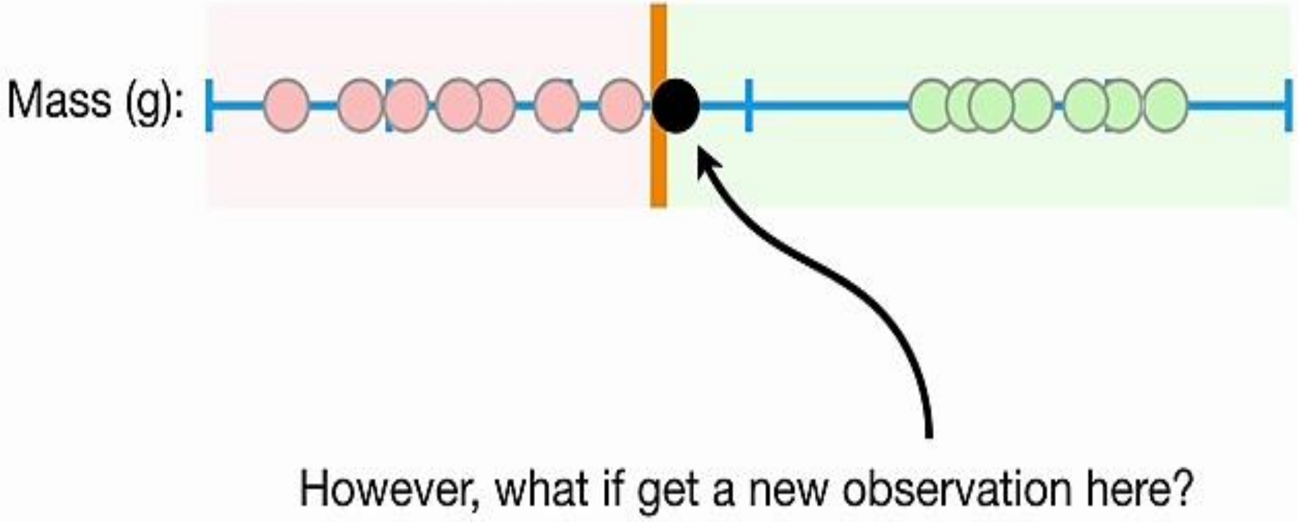


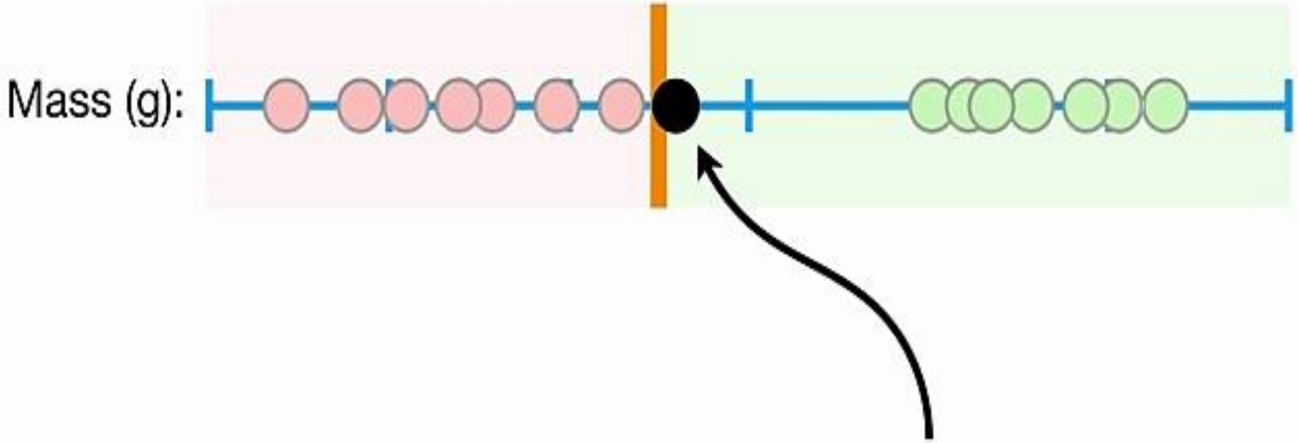
…we can classify it as ***not obese***.


Mass (g):
And when we get a new observation with more mass
than the threshold...

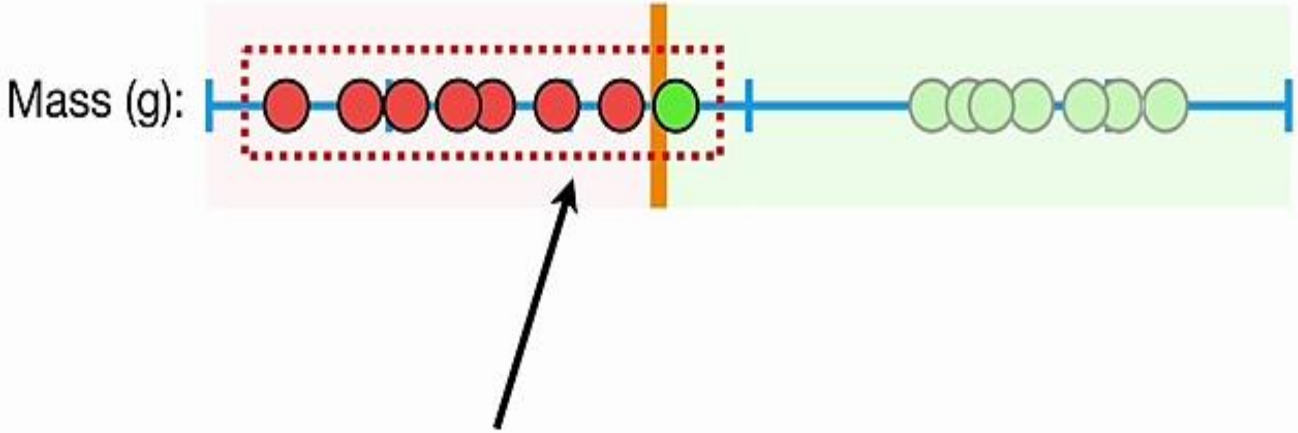


…we can classify it as ***obese***.

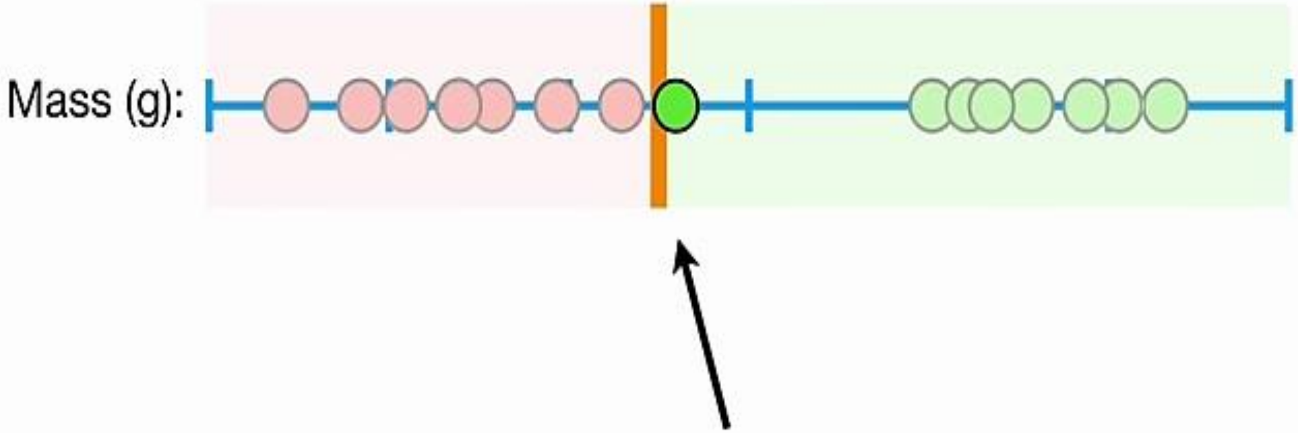

Mass (g):
However, what if get a new observation here?



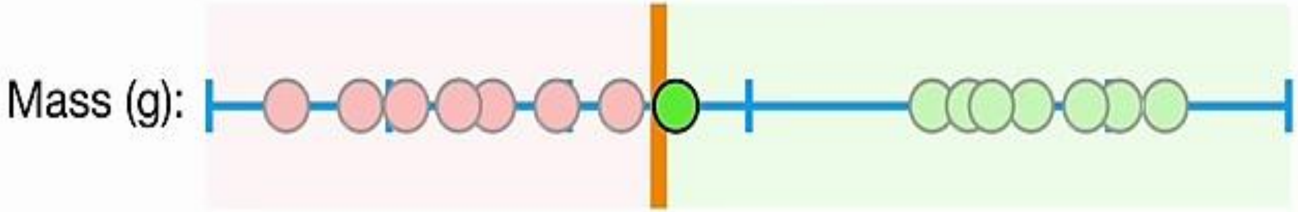
Because this observation has more mass than the threshold, we classify it as ***obese***.



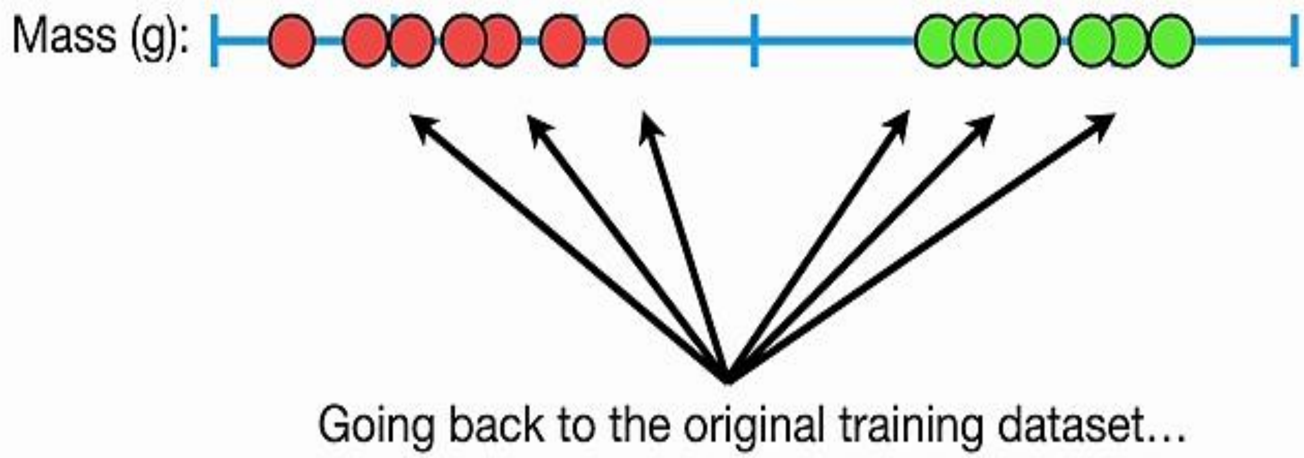
But that doesn't make sense, because it is much closer to the observations that are ***not obese***.

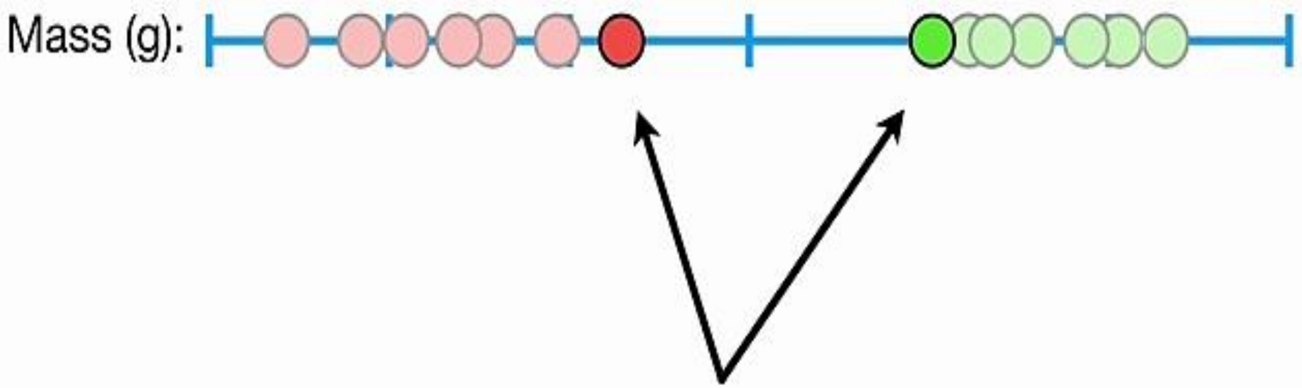


So this threshold is pretty lame.

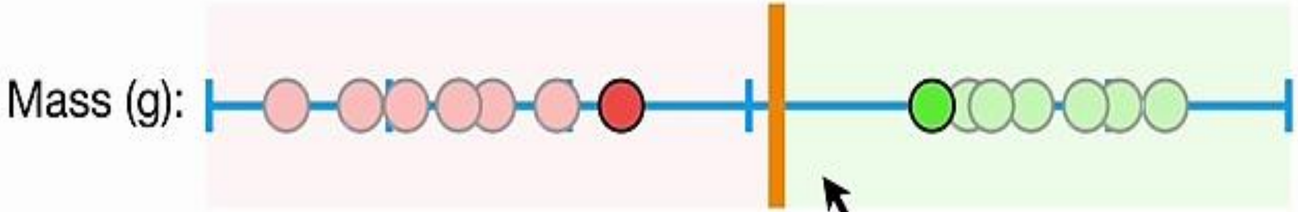


Can we do better?

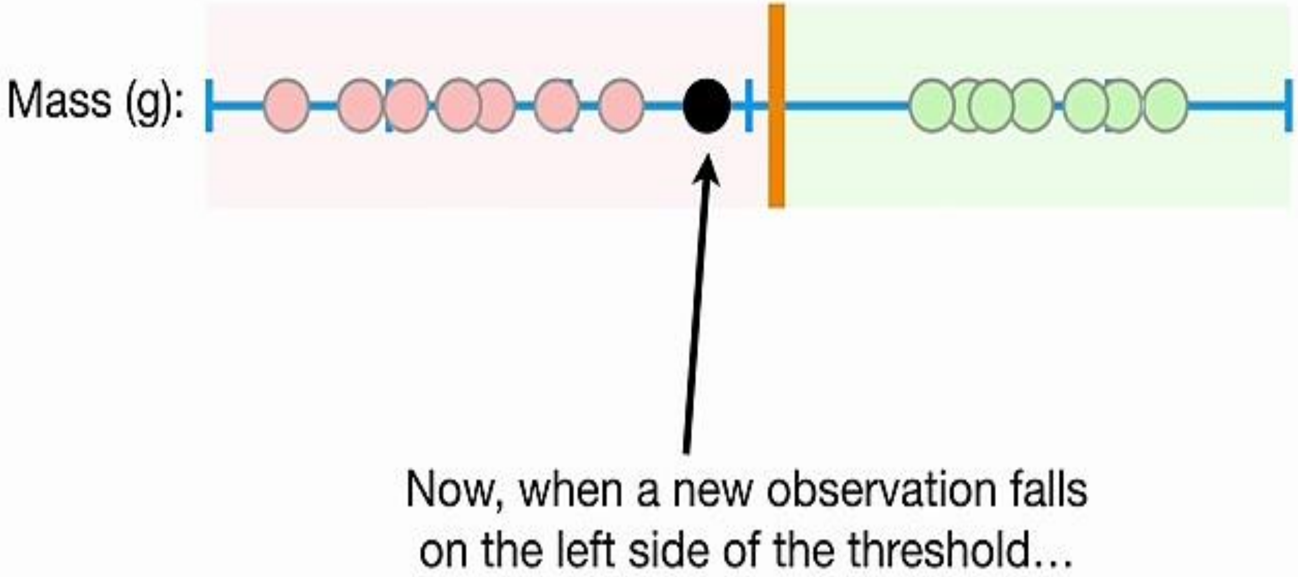

Mass (g):
Going back to the original training dataset...

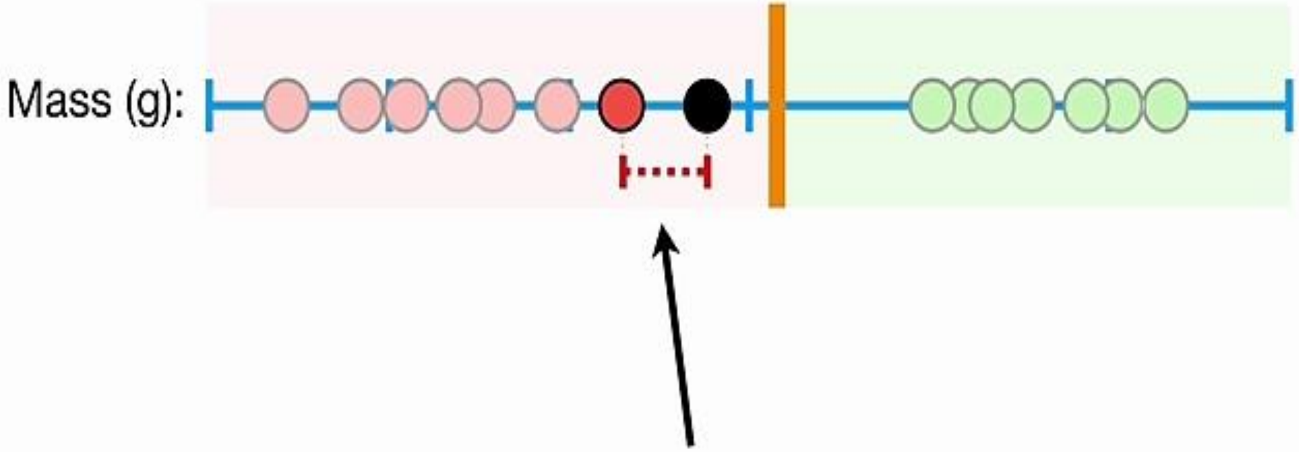


...we can focus on the observations on the edges of each cluster...

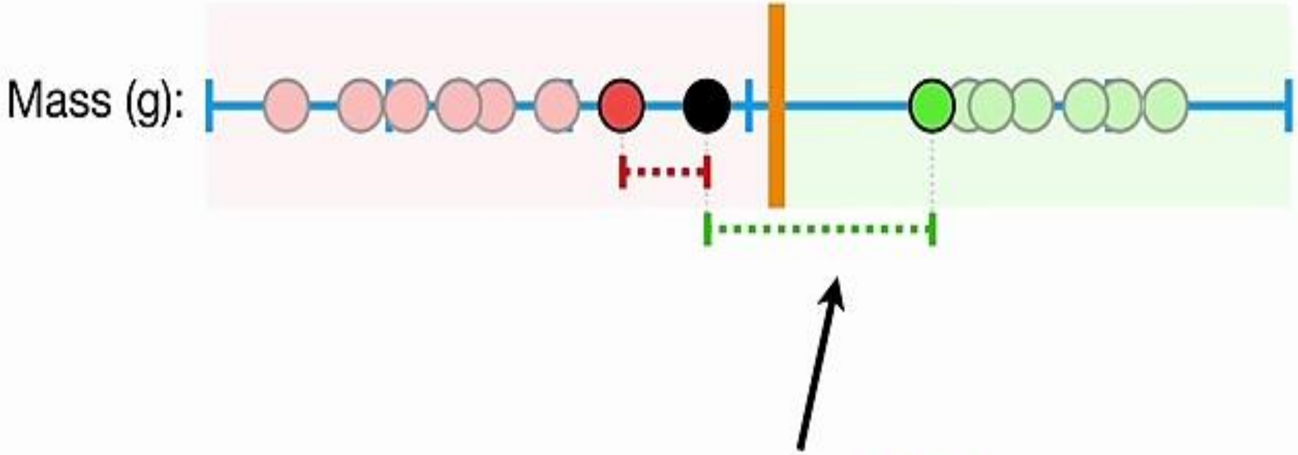


...and use the midpoint between
them as the threshold.

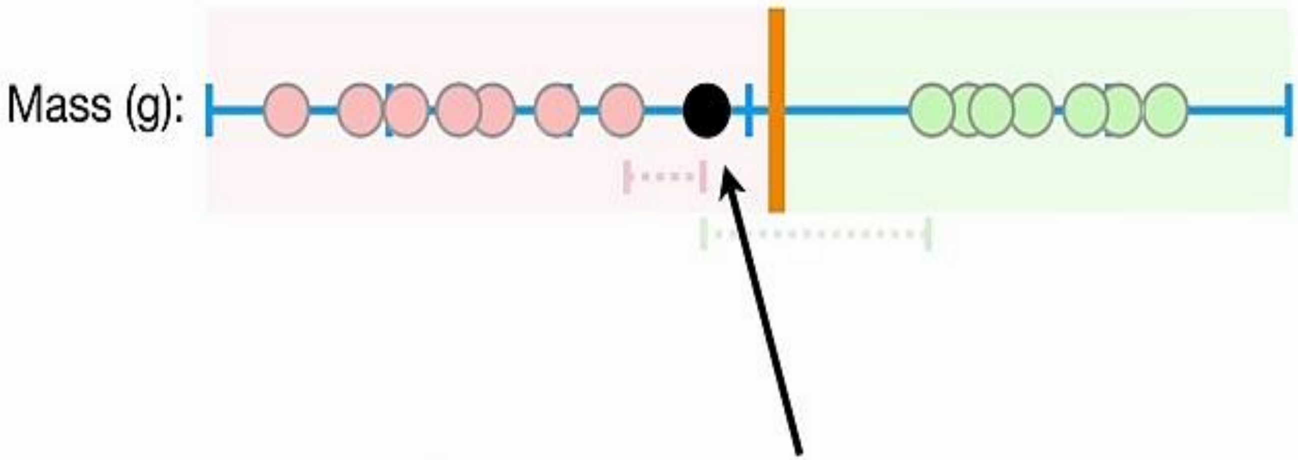

Mass (g):
Now, when a new observation falls on the left side of the threshold...



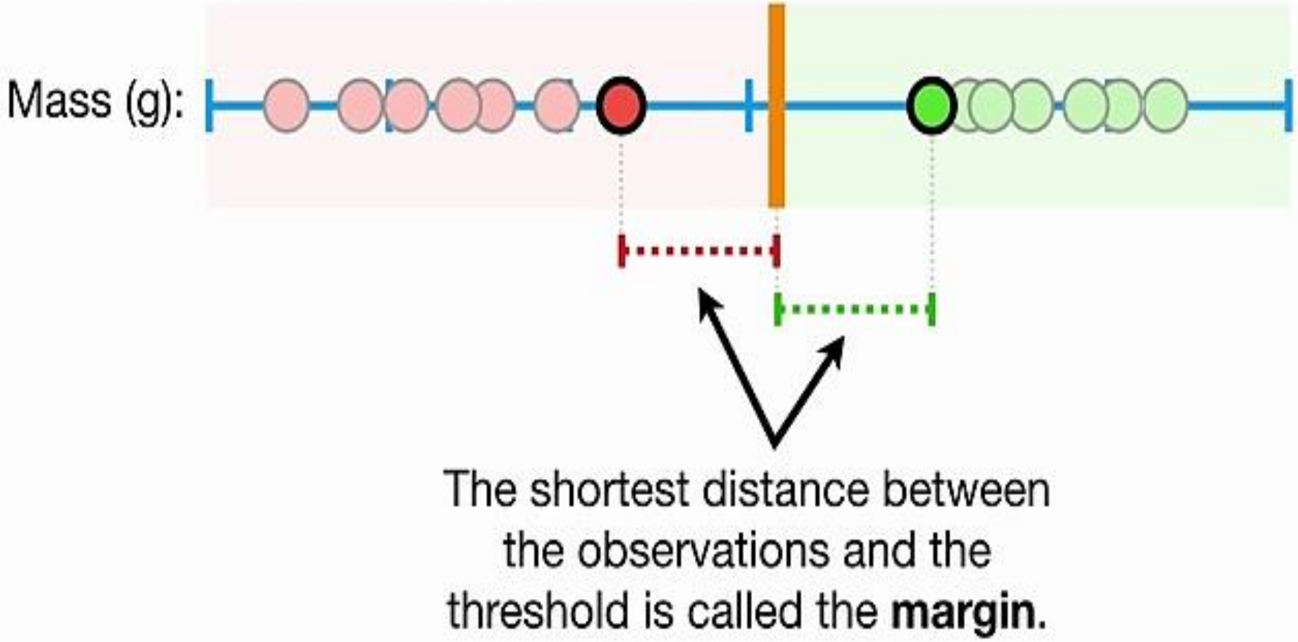
...it will be closer to the observations that are ***not obese***...

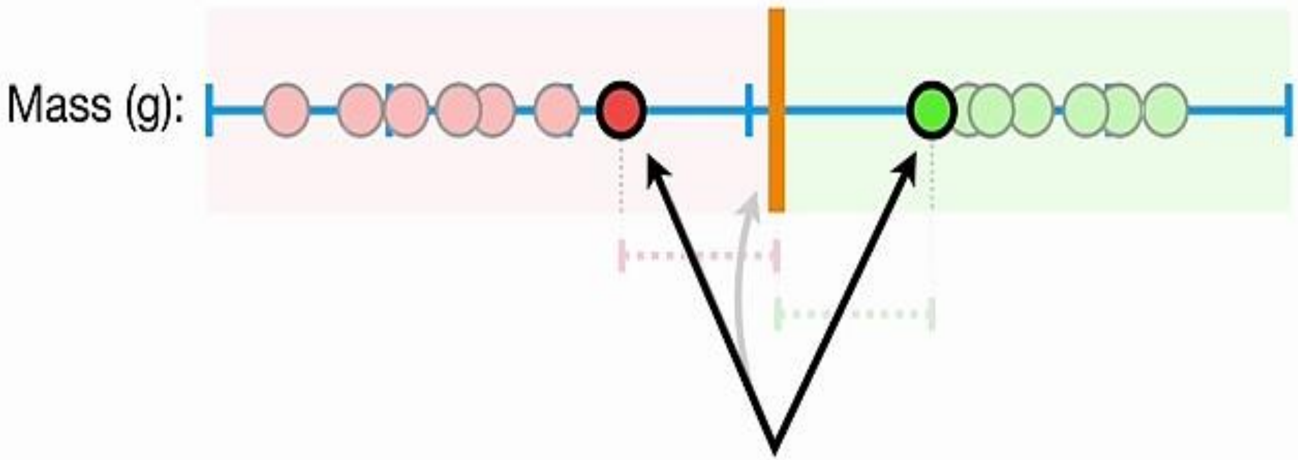


…than it is to the ***obese*** observations.

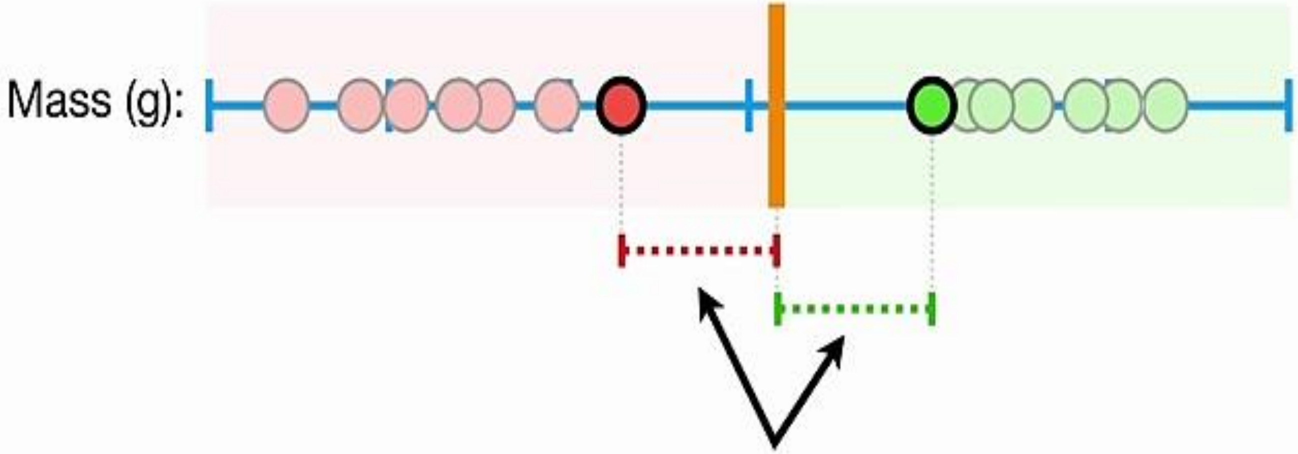


So it makes sense to classify this new observation as ***not obese***.

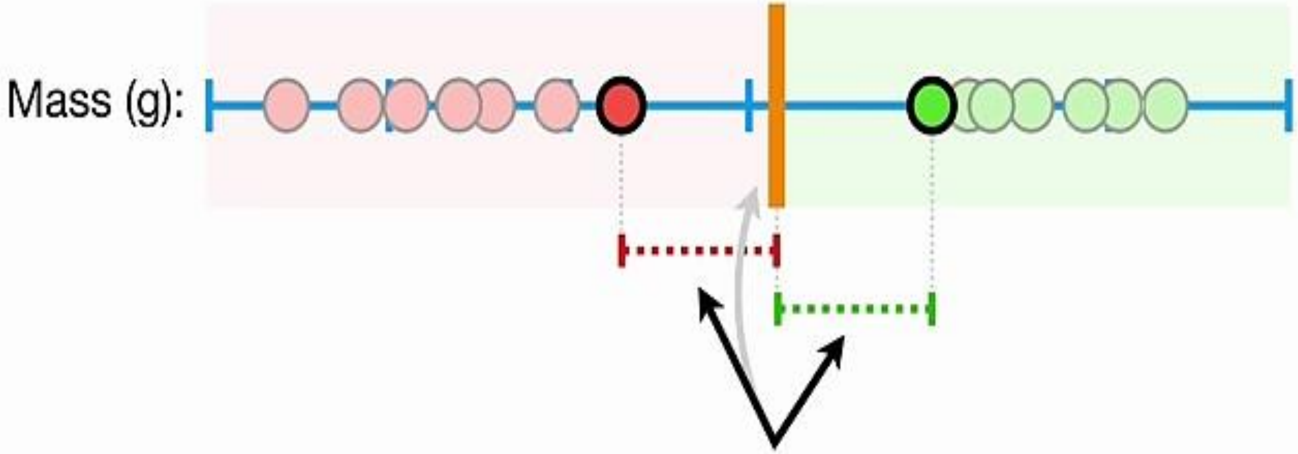

Mass (g):
The shortest distance between
the observations and the
threshold is called the **margin.**



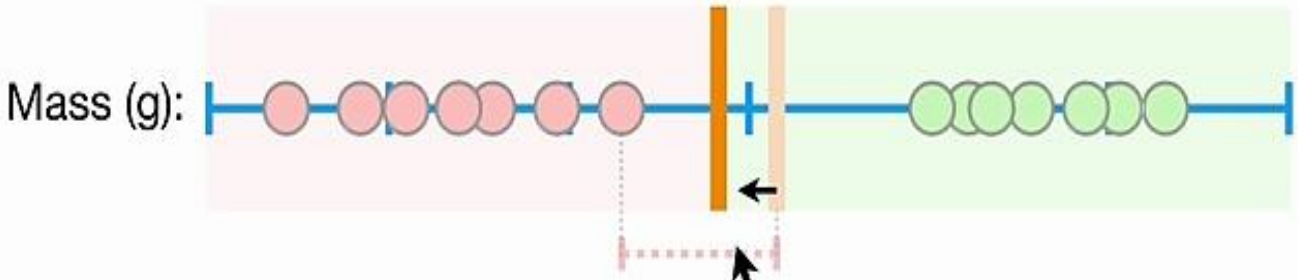
Since we put the threshold halfway between these two observations...



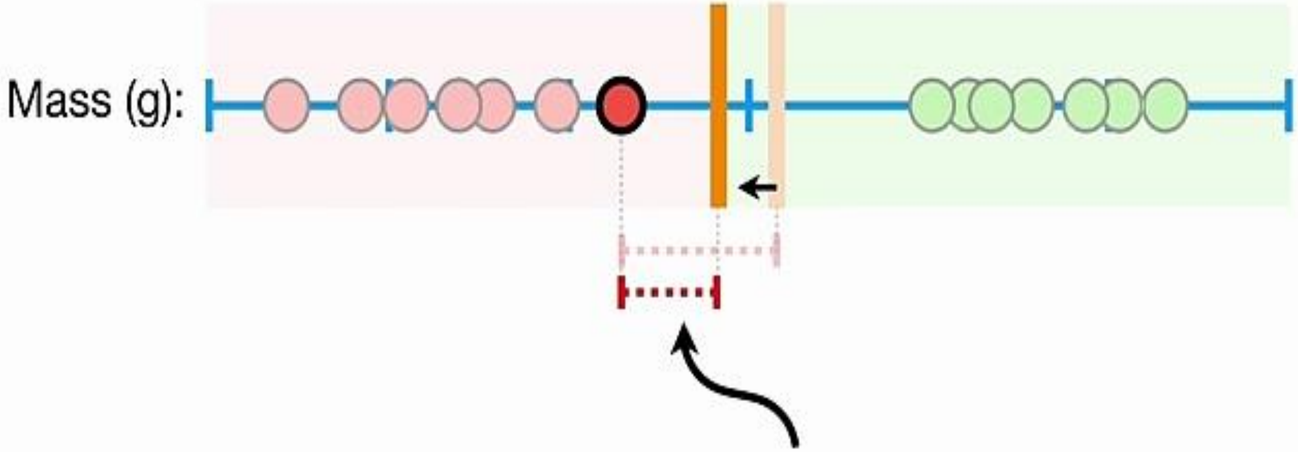
...the distances between the observations and the threshold are the same and both reflect the **margin**.



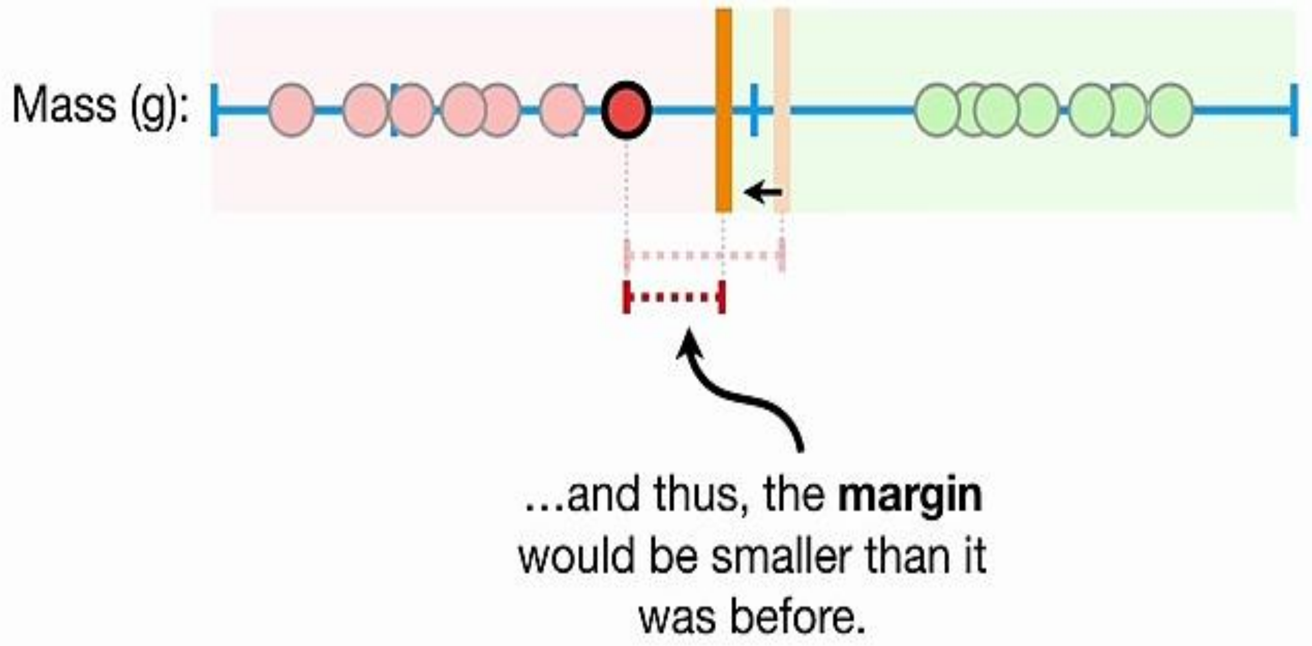
When the threshold is halfway between the two observations, the **margin** is as large as it can be.

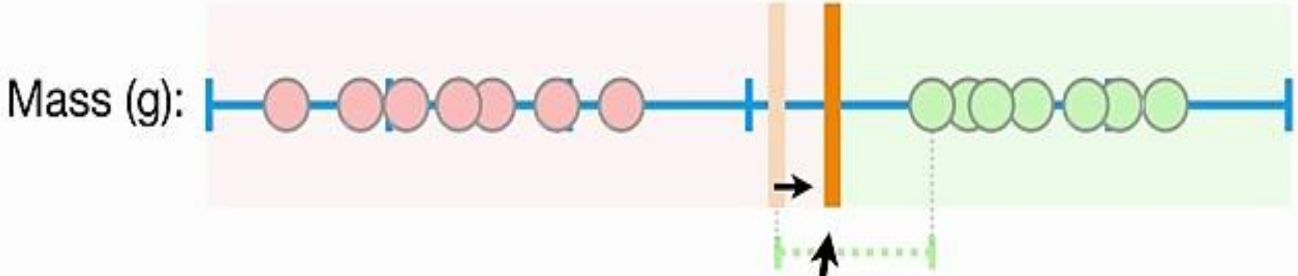


For example, if we moved the threshold to the left a little bit…

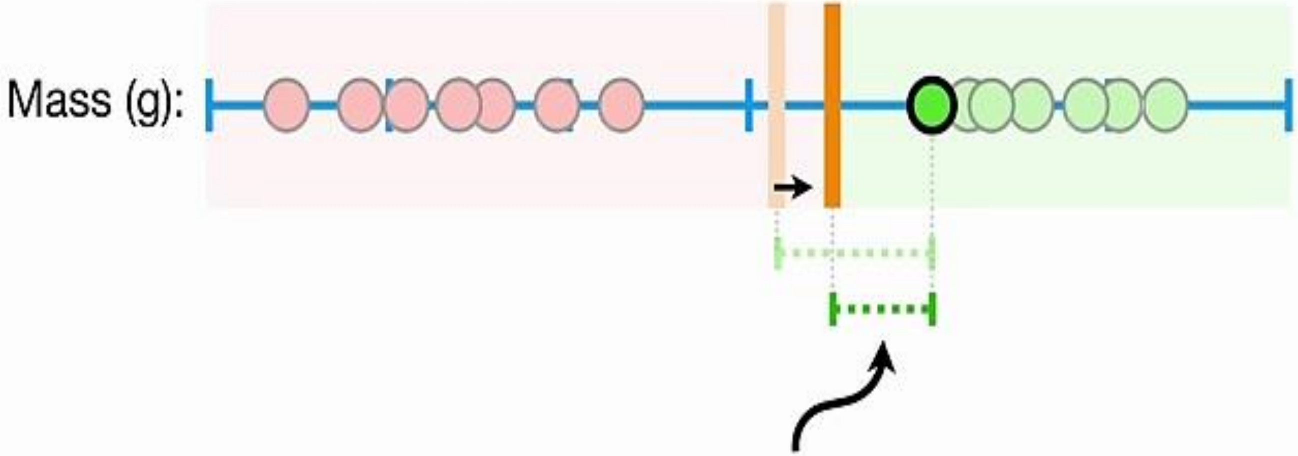


...then the distance between the threshold and the observation that is ***not obese*** would be smaller...

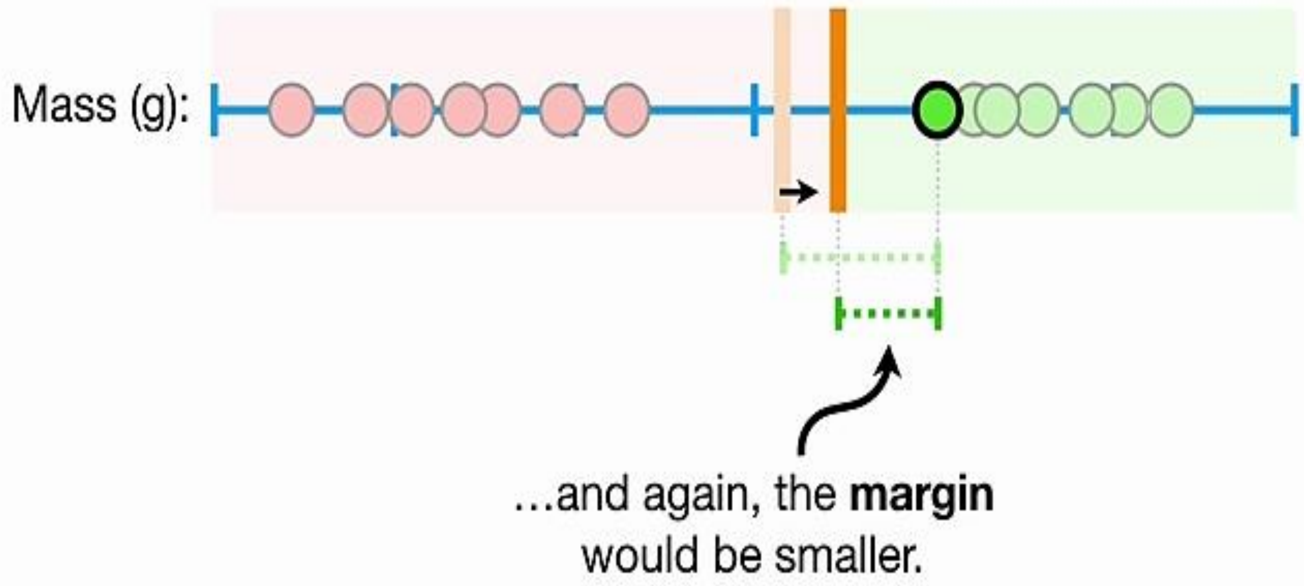

Mass (g):
...and thus, the **margin** would be smaller than it was before.

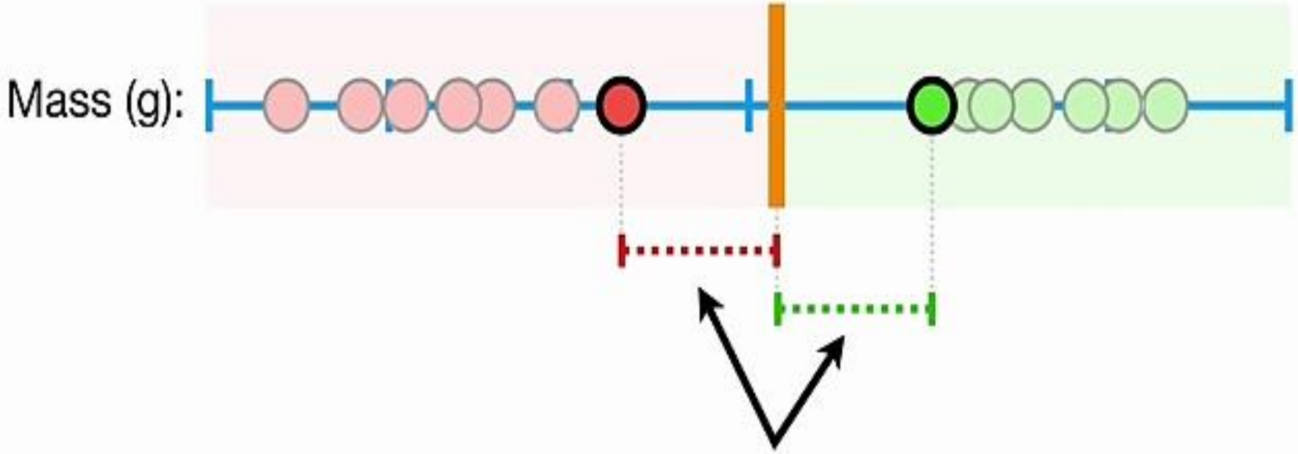


And if we moved the threshold to
the right a little bit…

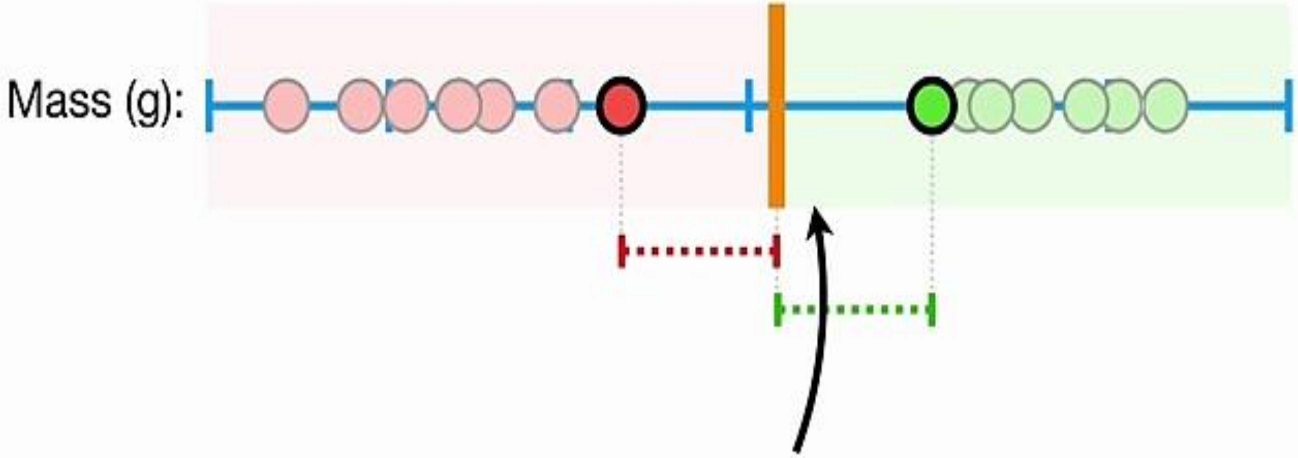


...then the distance between the *obese* observation and the threshold would get smaller...


Mass (g):
…and again, the **margin**
would be smaller.

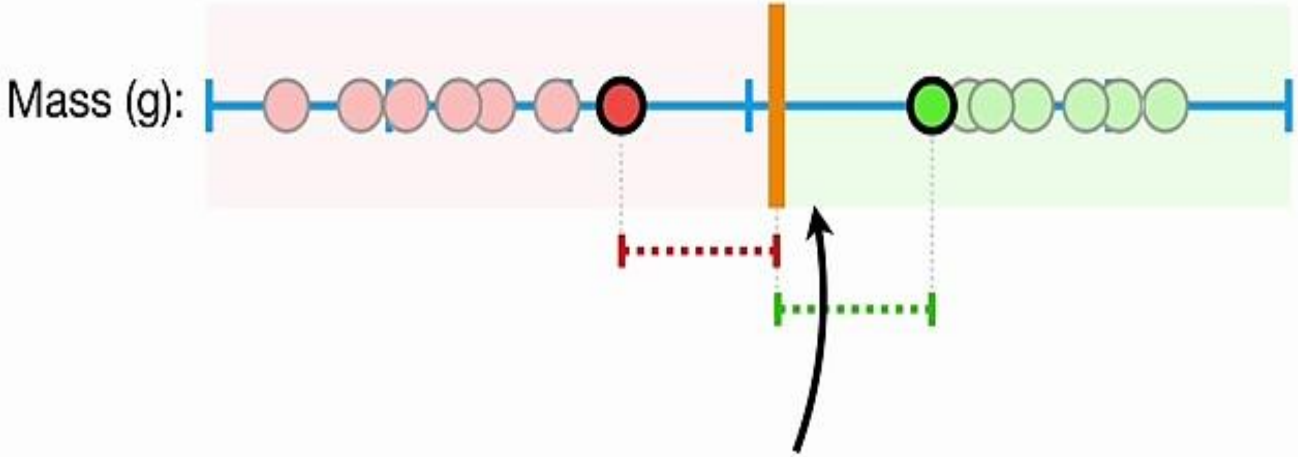


When we use the threshold that gives us the largest **margin** to make classifications…

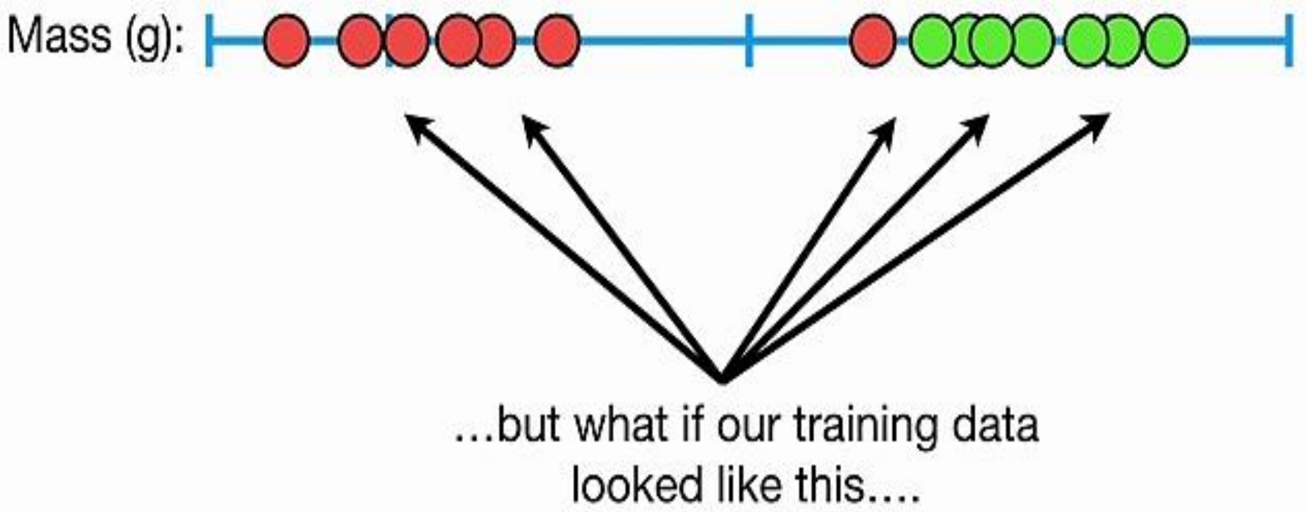


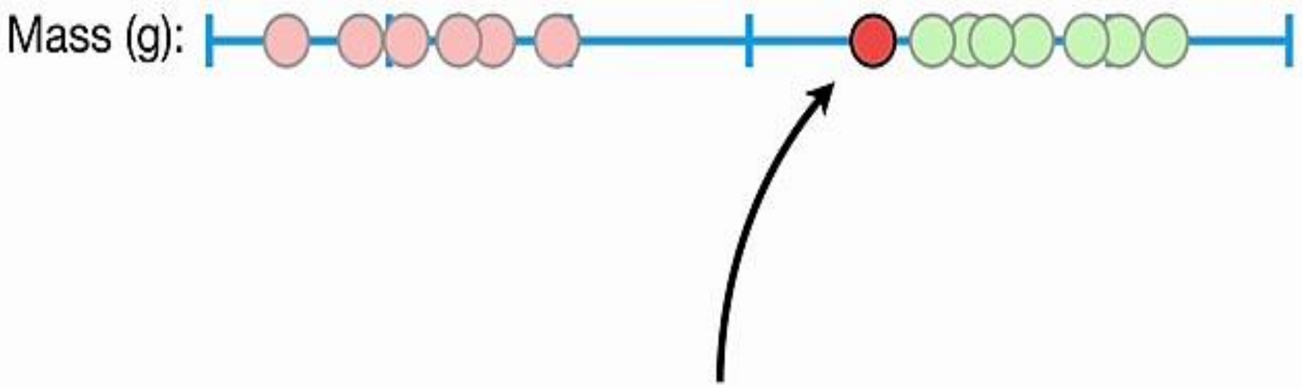
…we are using a

Maximal Margin Classifier.

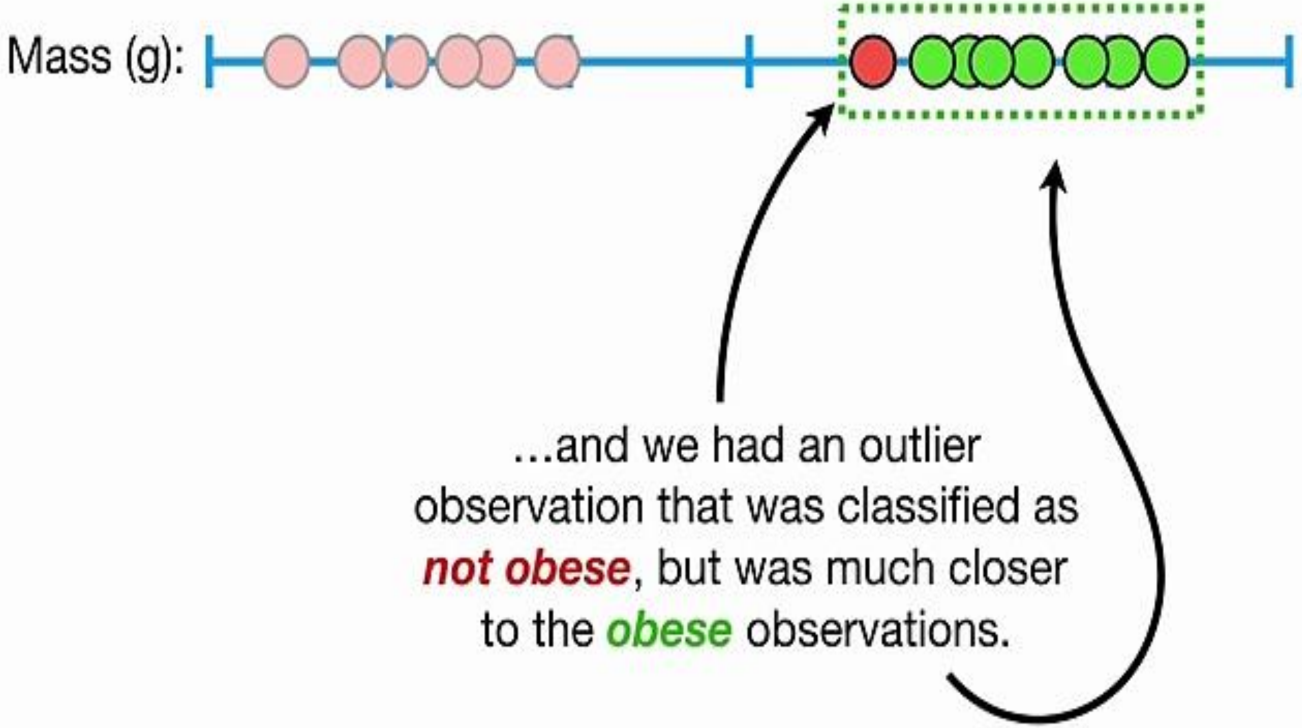


Maximal Margin Classifiers
seem pretty cool...

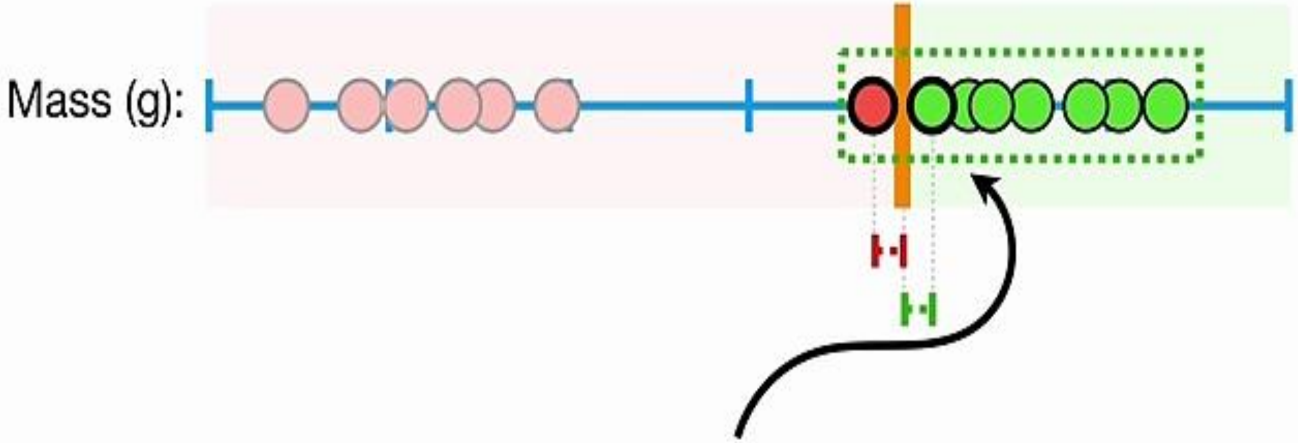

Mass (g):
...but what if our training data
looked like this....



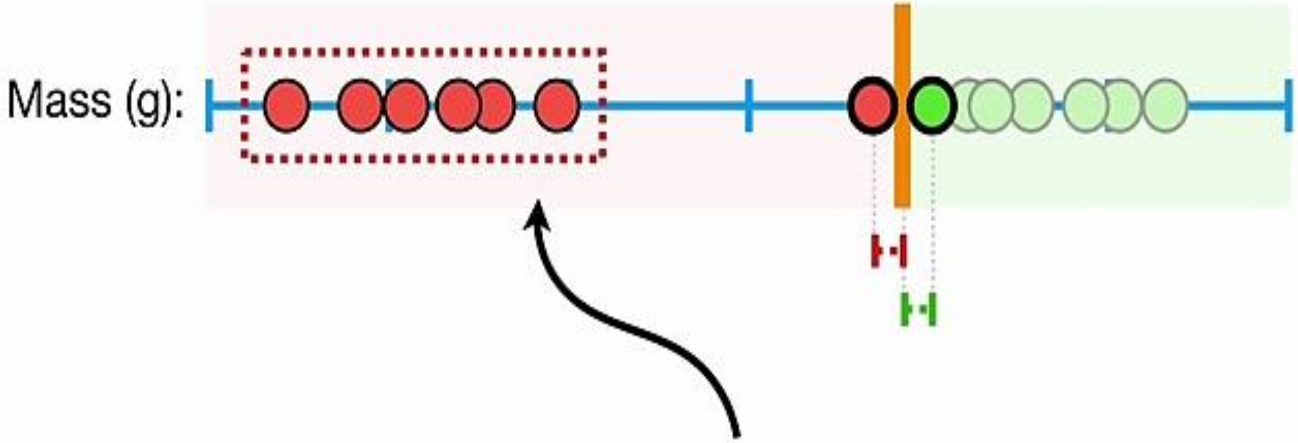
...and we had an outlier observation that was classified as ***not obese***, but was much closer to the ***obese*** observations.



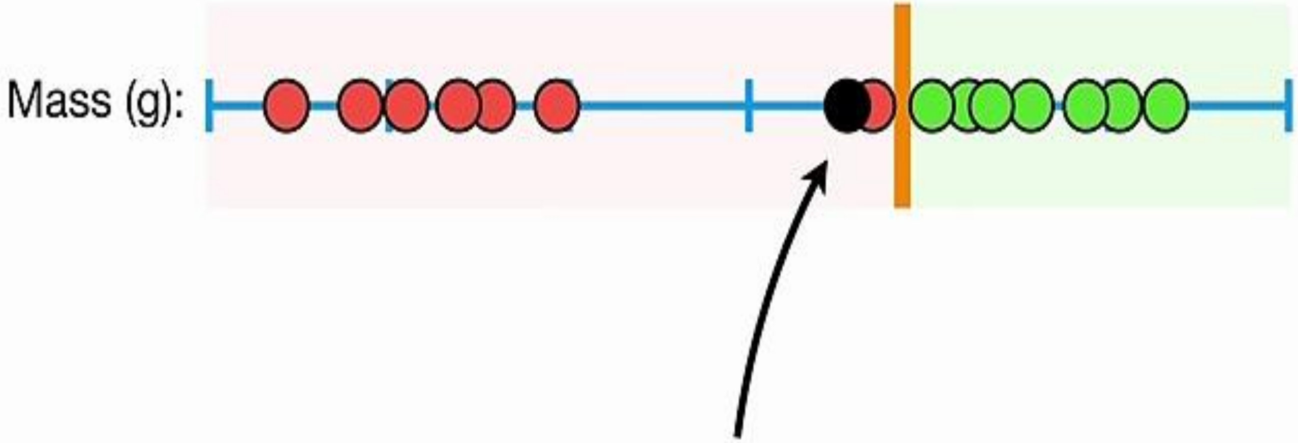
Mass (g):
...and we had an outlier observation that was classified as *not obese*, but was much closer to the *obese* observations.



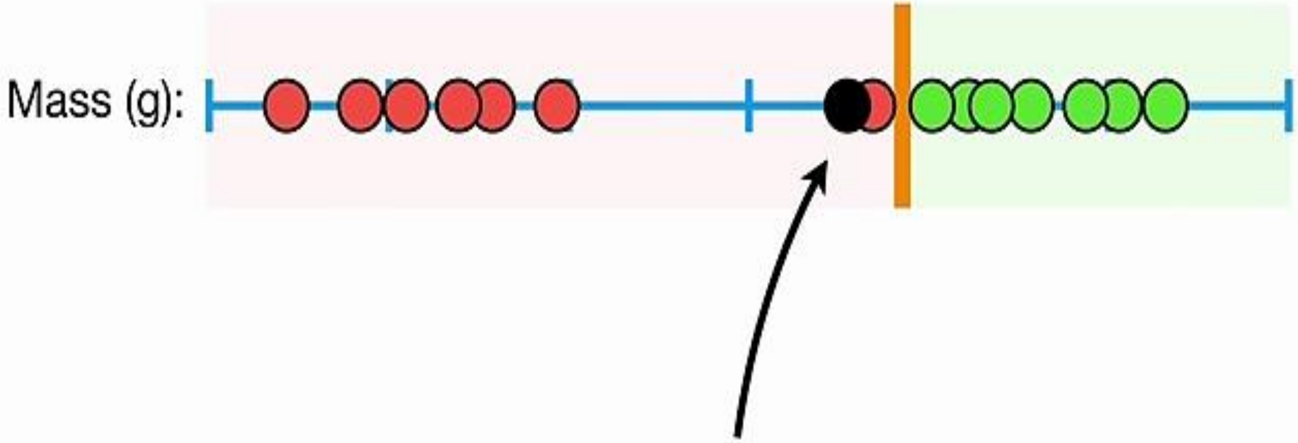
In this case, the **Maximum Margin Classifier** would be super close to the ***obese*** observations...



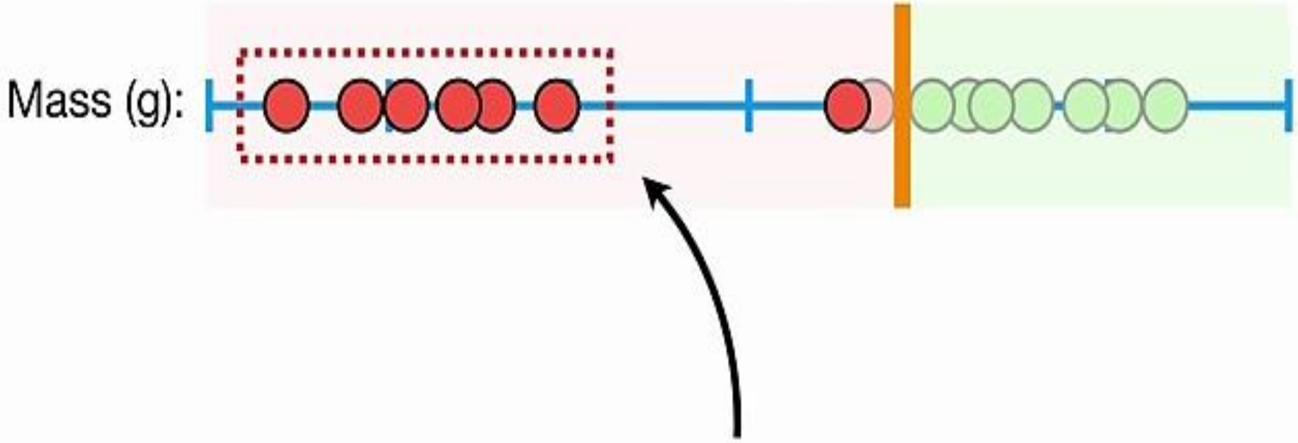
...and really far from the majority of the observations that are **not obese**.



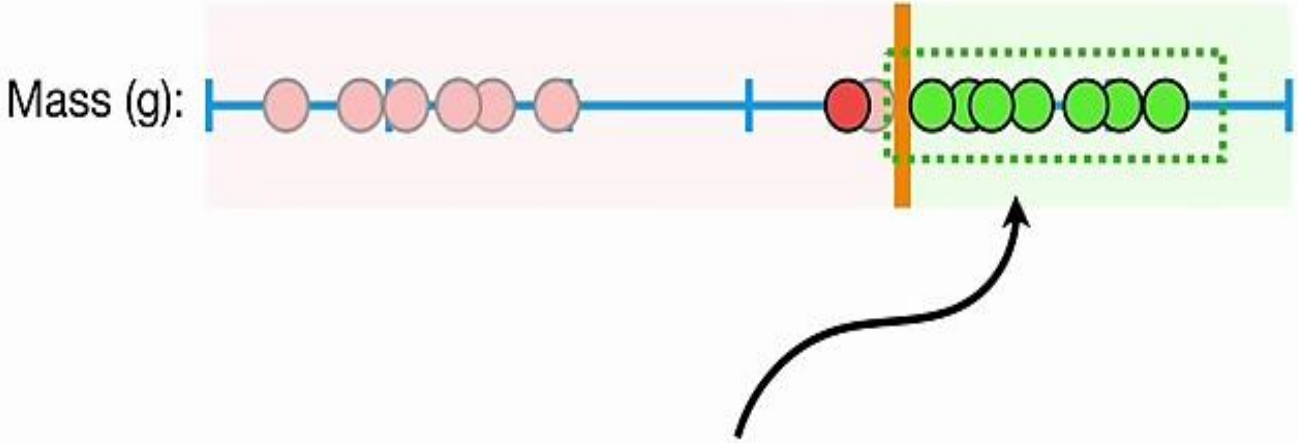
Now, if we got this new observation…



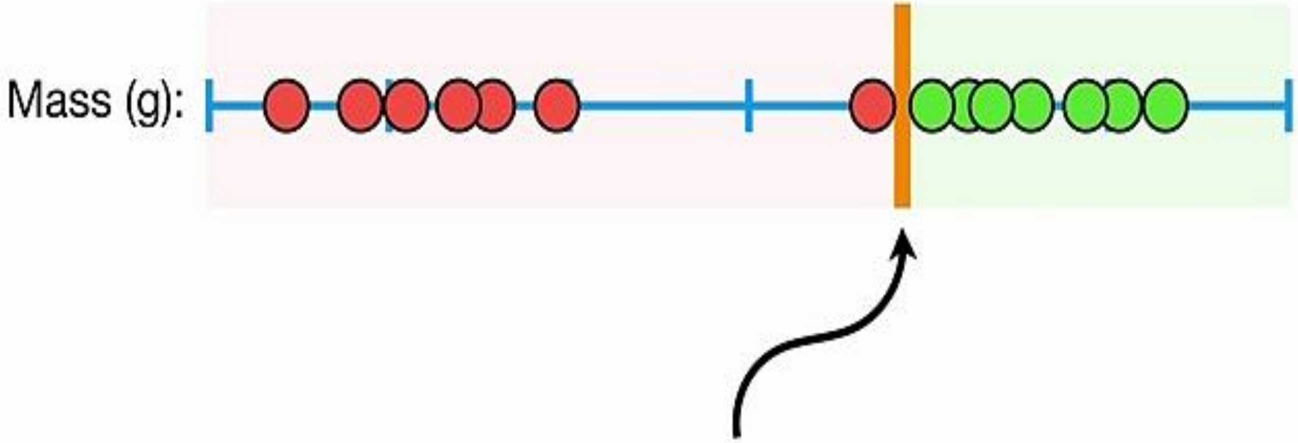
…we would classify it as ***not obese***, even though most of the ***not obese*** observations are much further away than the ***obese*** observations.



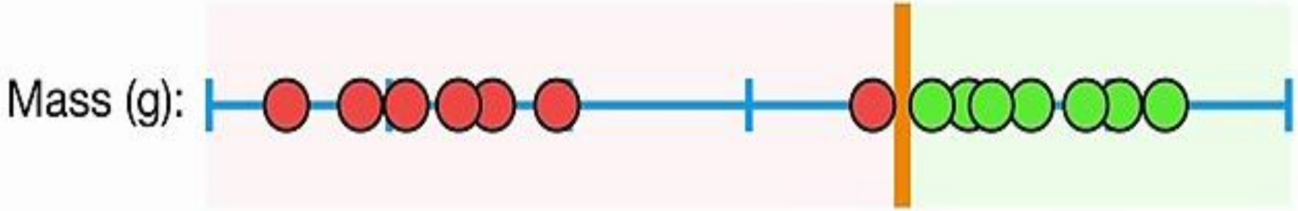
...we would classify it as ***not obese***, even though most of the ***not obese*** observations are much further away than the ***obese*** observations.



...we would classify it as ***not obese***, even though most of the ***not obese*** observations are much further away than the ***obese*** observations.

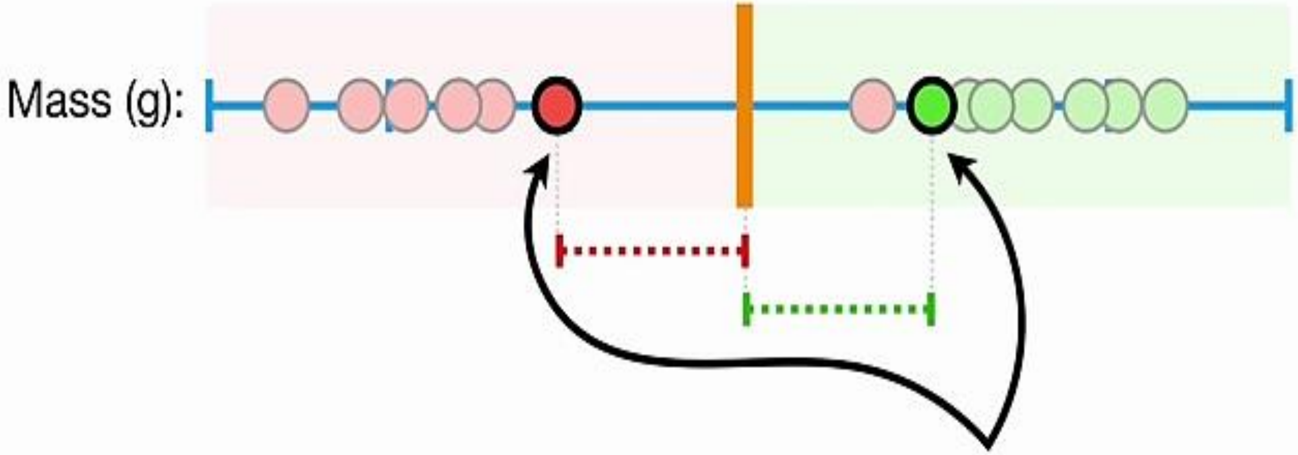


So **Maximal Margin Classifiers** are *super sensitive to outliers* in the training data and that makes them pretty lame.

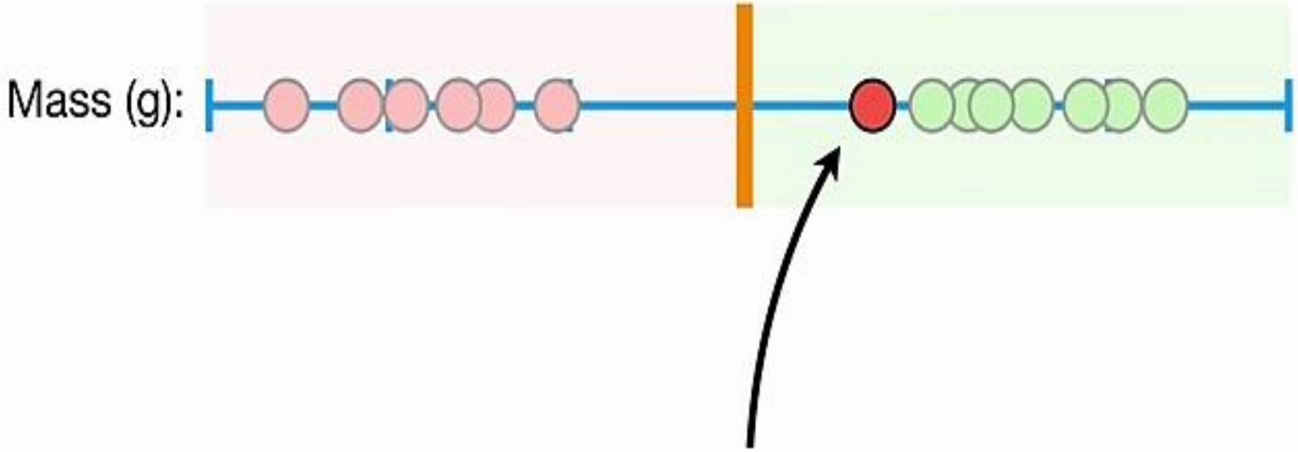


Can we do better?

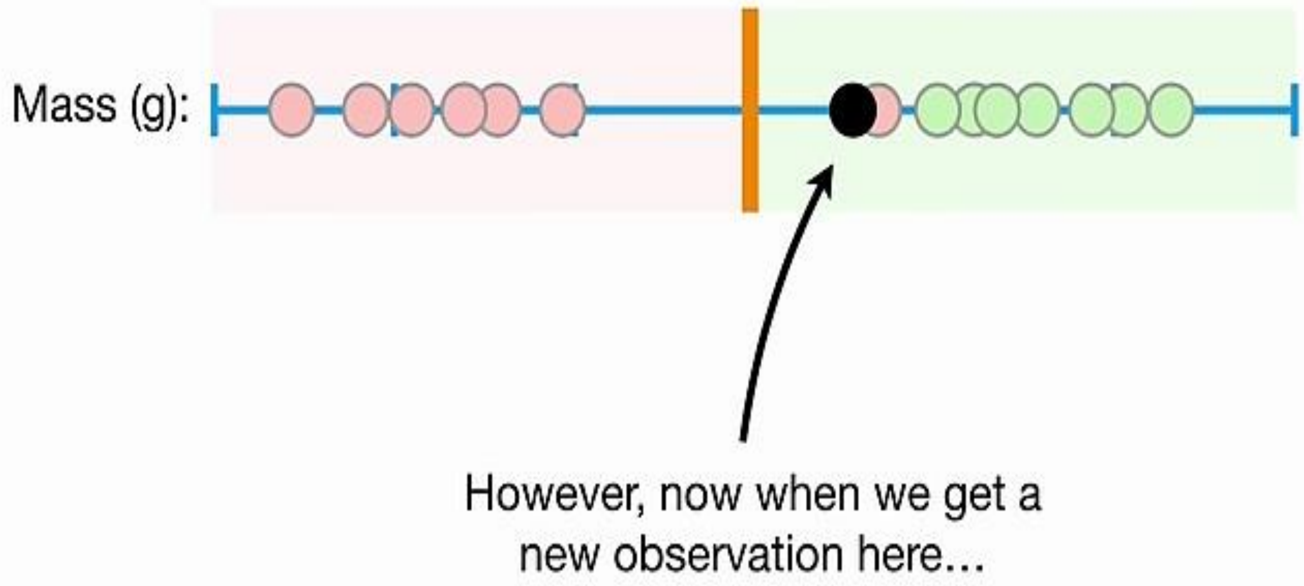
To make a threshold that is not so sensitive to outliers we must **allow misclassifications**.

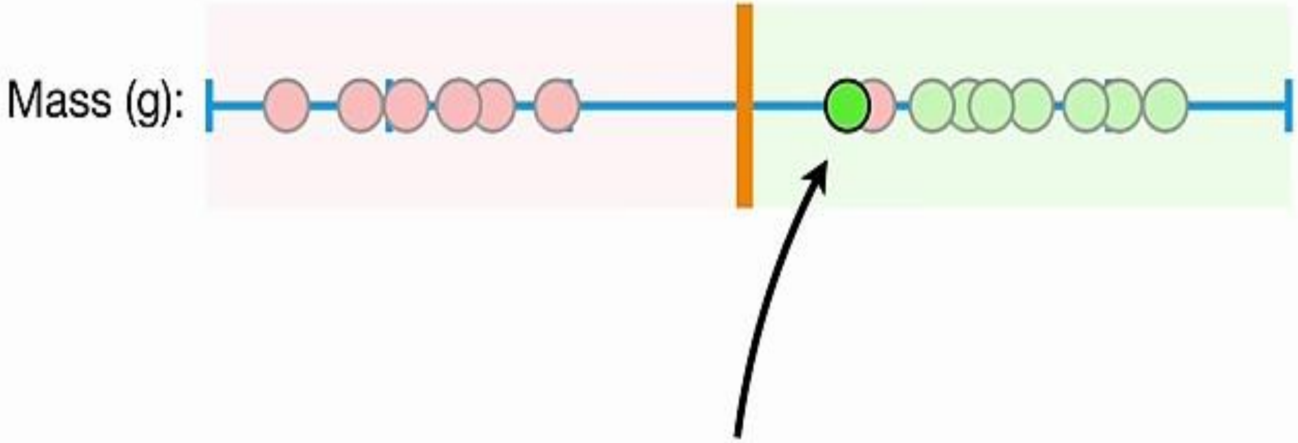


For example, if we put the threshold halfway between these two observations…

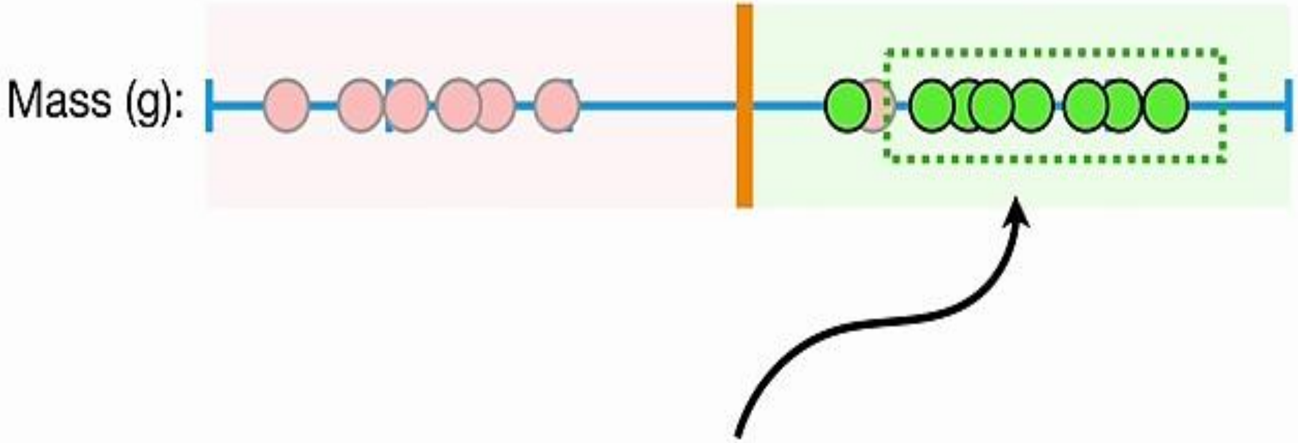


…then we will misclassify this observation.

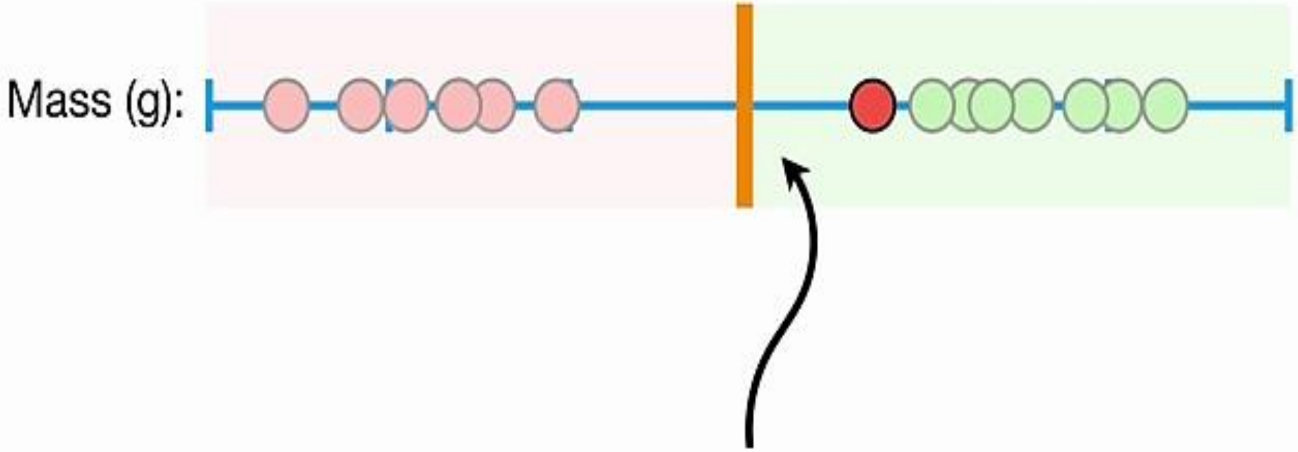

Mass (g):
However, now when we get a
new observation here...



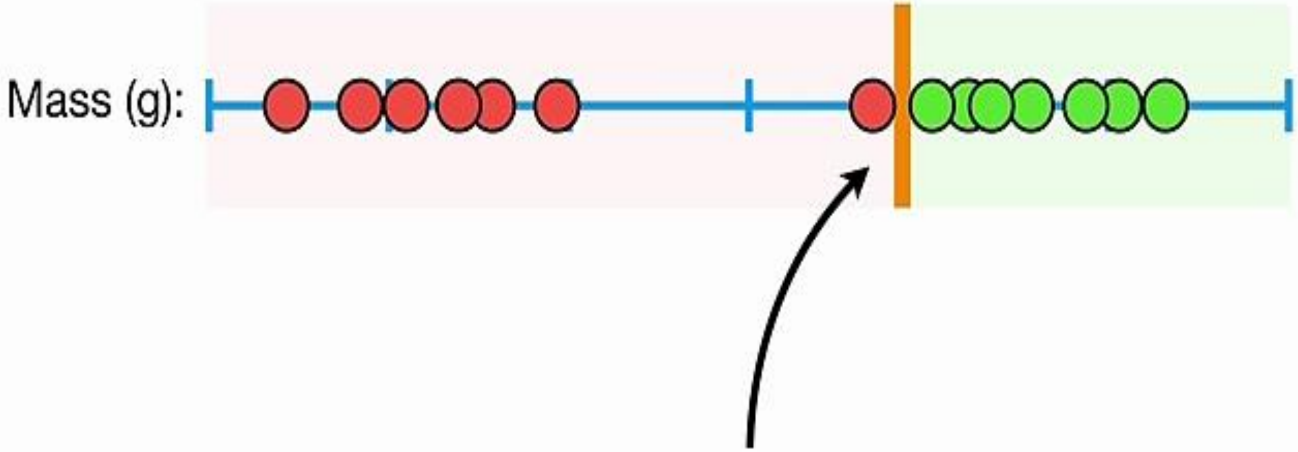
...we will classify it as ***obese***...



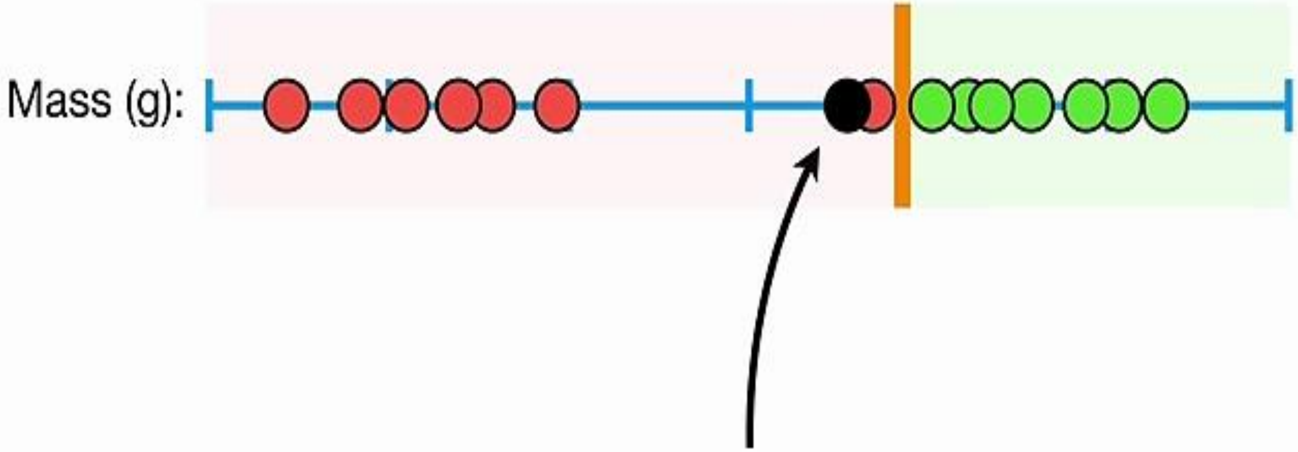
...and that makes sense because it is closer to most of the ***obese*** observations.



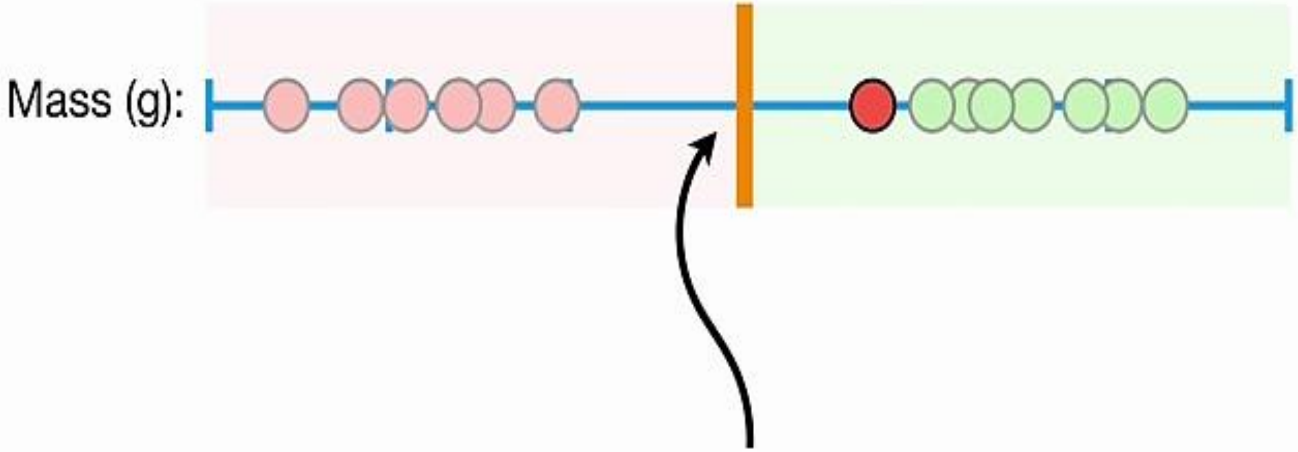
Choosing a threshold that allows misclassifications is an example of the **Bias/Variance Tradeoff** that plagues all of machine learning.



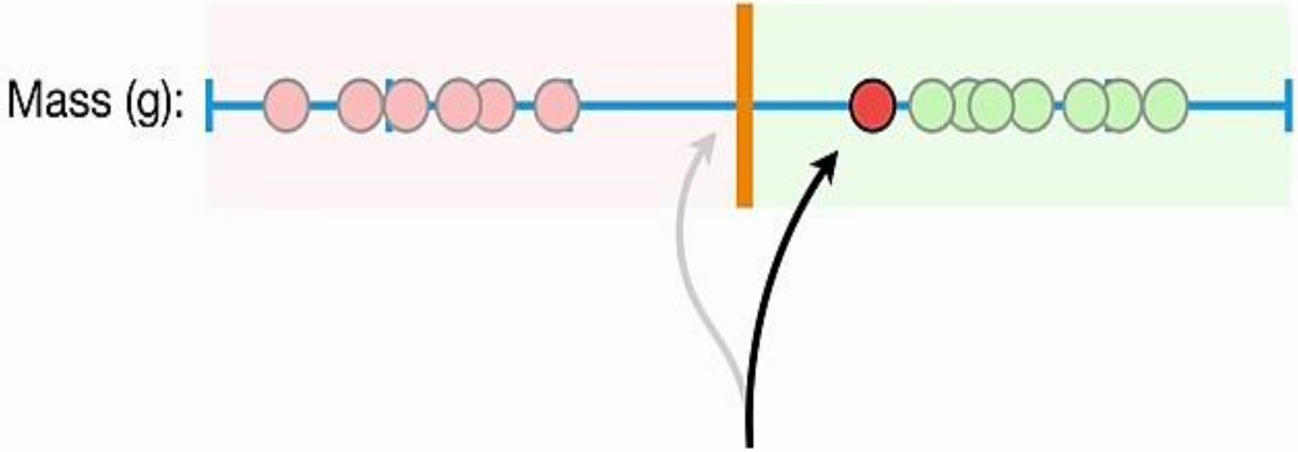
In other words, before we allowed misclassifications, we picked a threshold that was very sensitive to the training data (low bias)…



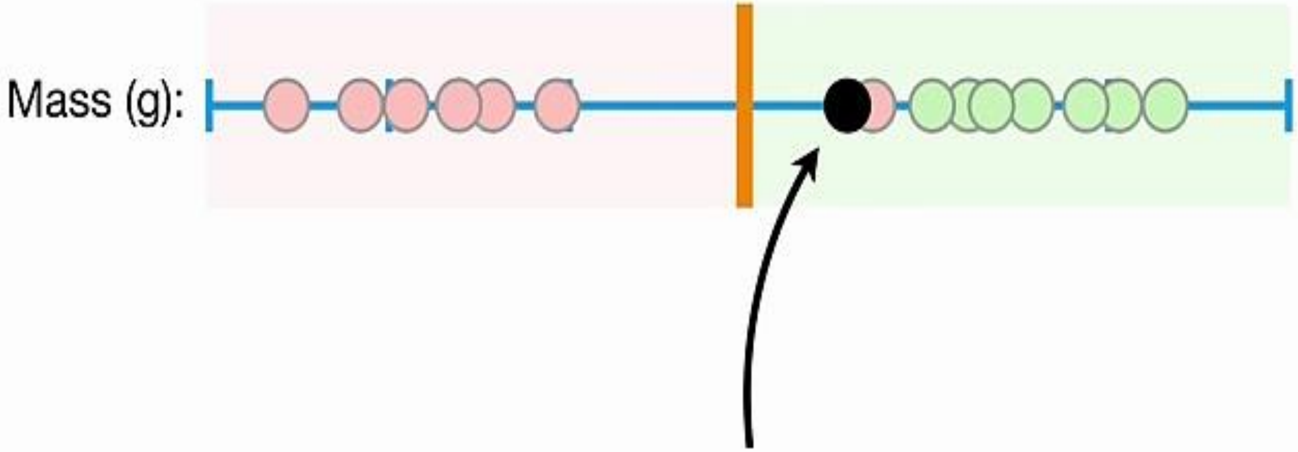
...and it performed poorly when
we got new data (high variance).



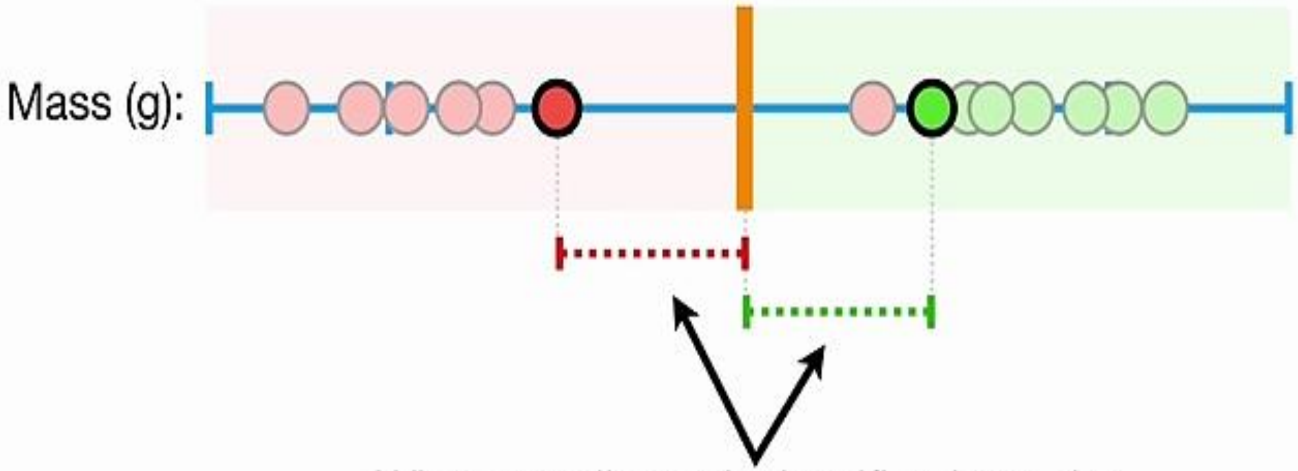
In contrast, when we picked a threshold that was less sensitive to the training data and allowed misclassifications (higher bias)…



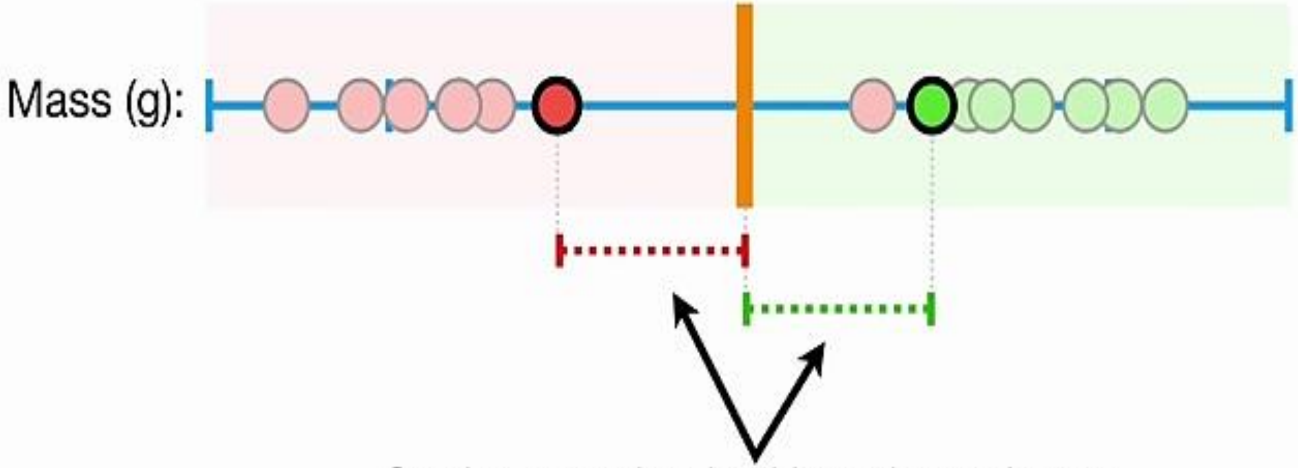
In contrast, when we picked a threshold that was less sensitive to the training data and allowed misclassifications (higher bias)…



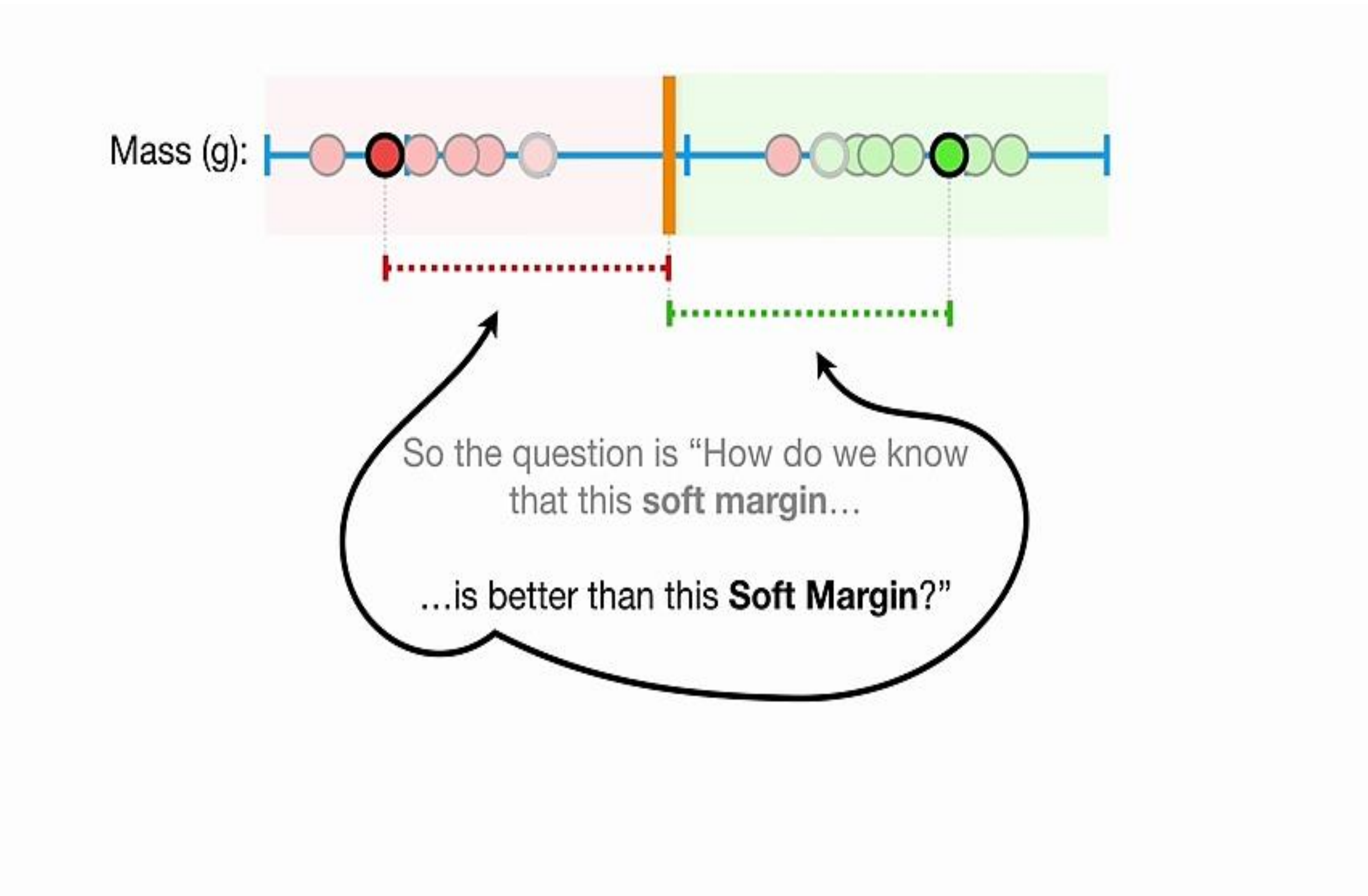
...it performed better when we got new data (low variance).

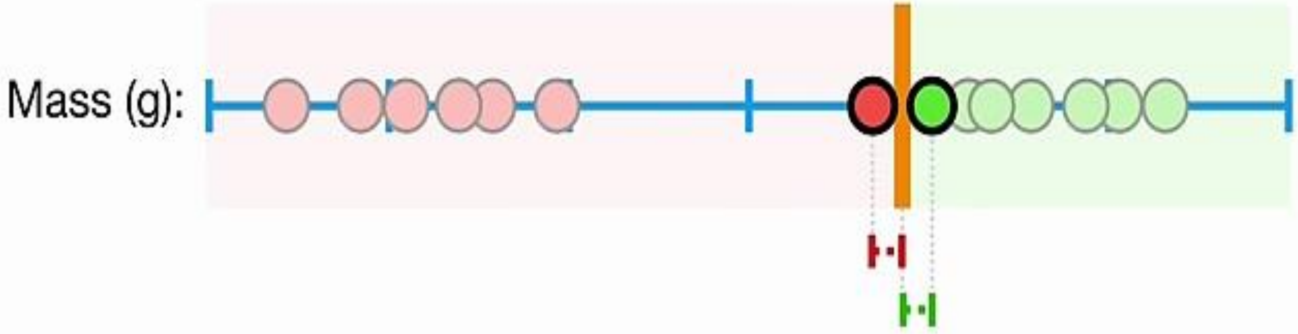


When we allow misclassifications, the distance between the observations and the threshold is called a **Soft Margin**.

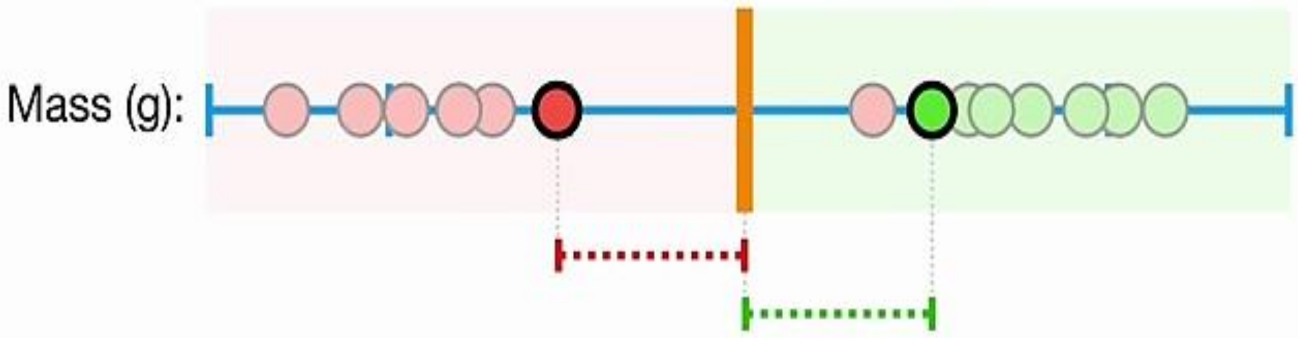


So the question is "How do we know that this **soft margin**...

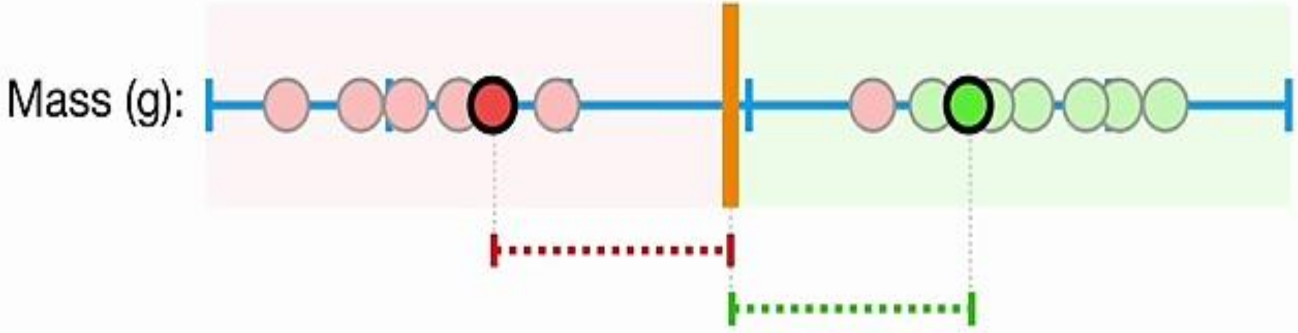

Mass (g):
So the question is "How do we know that this **soft margin**...
...is better than this **Soft Margin**?"



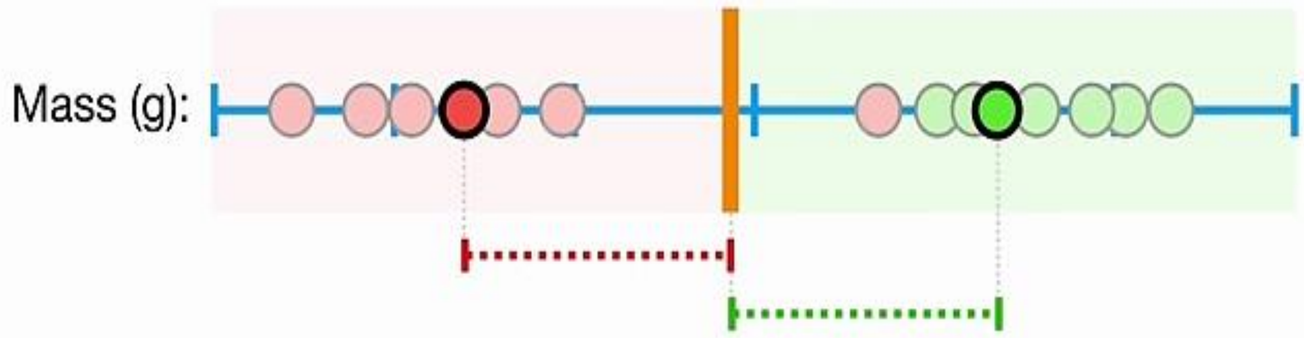
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



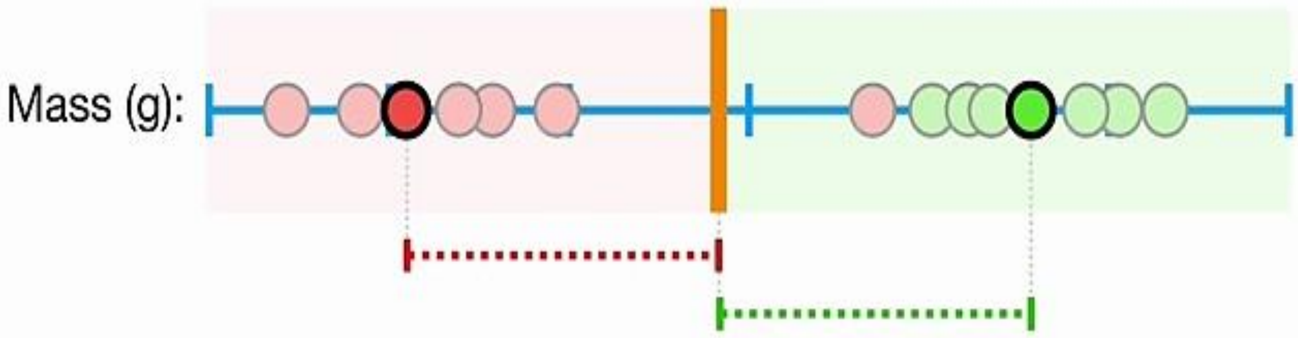
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



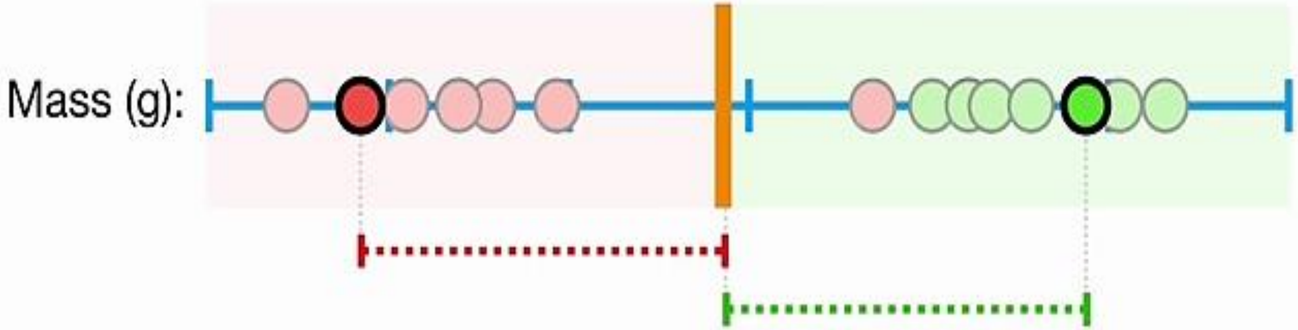
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



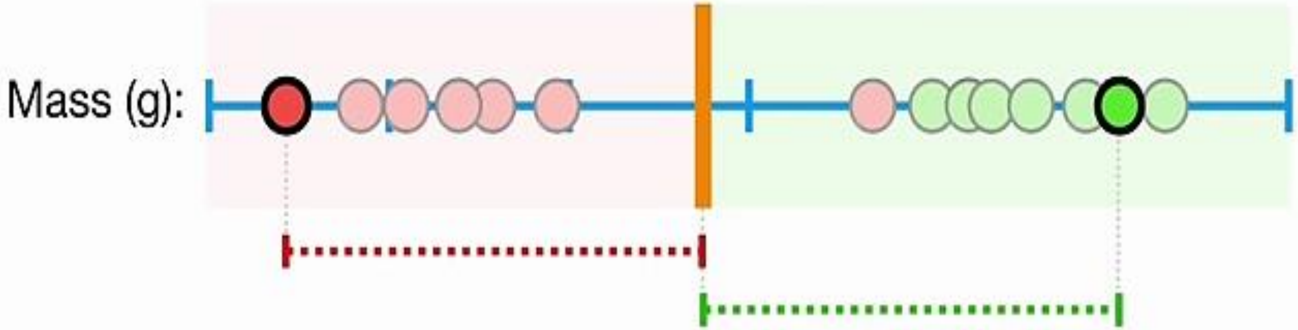
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



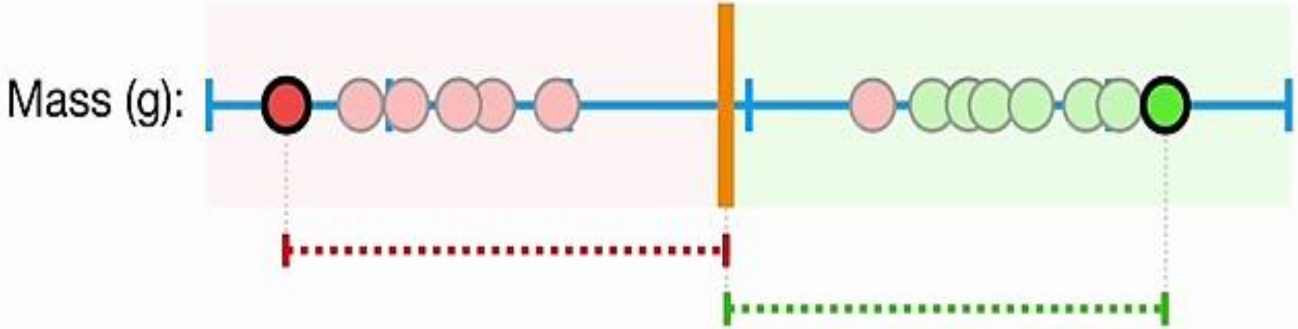
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



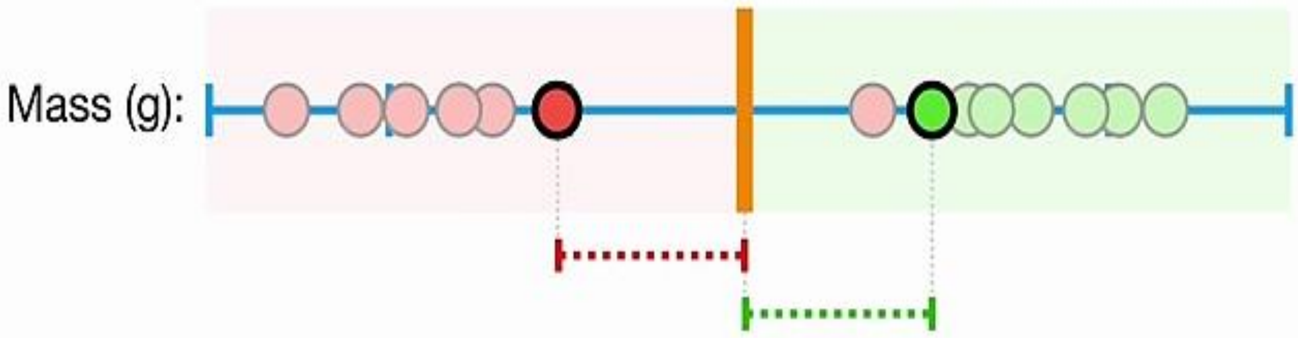
We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.



We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.

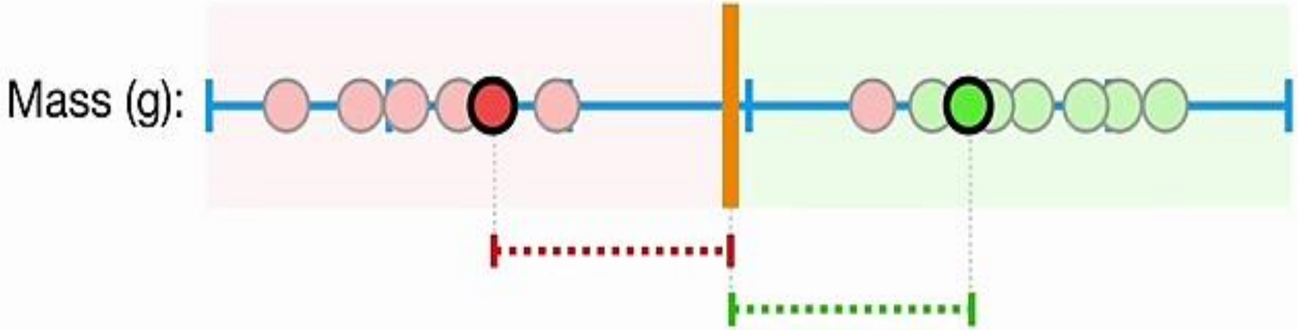


We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.

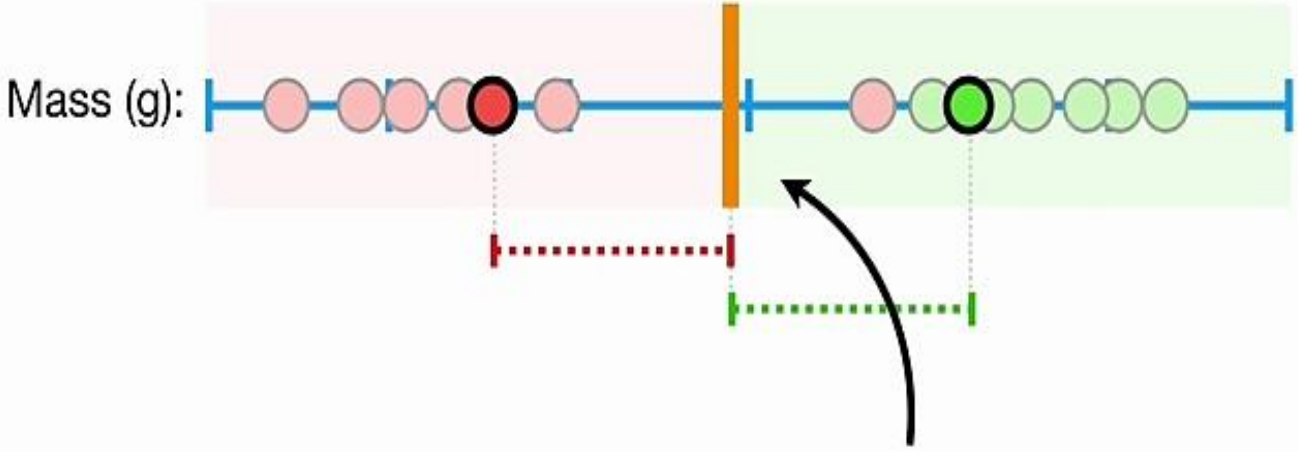


We can determine how many misclassifications and observations to allow inside of the **Soft Margin** to get the best classification.

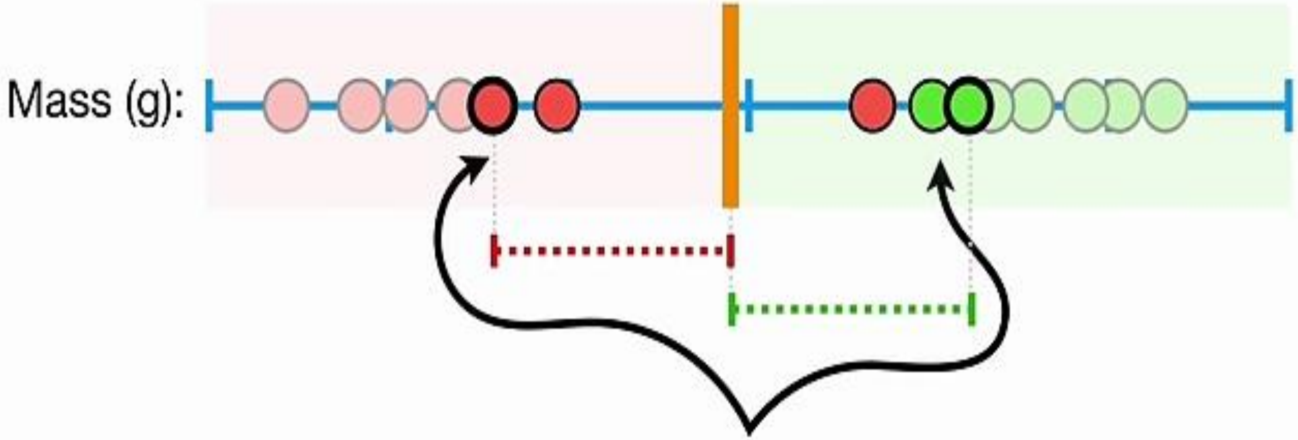
Ideally we should minimize the number of misclassification and the number of observation within the margin to avoid overfitting



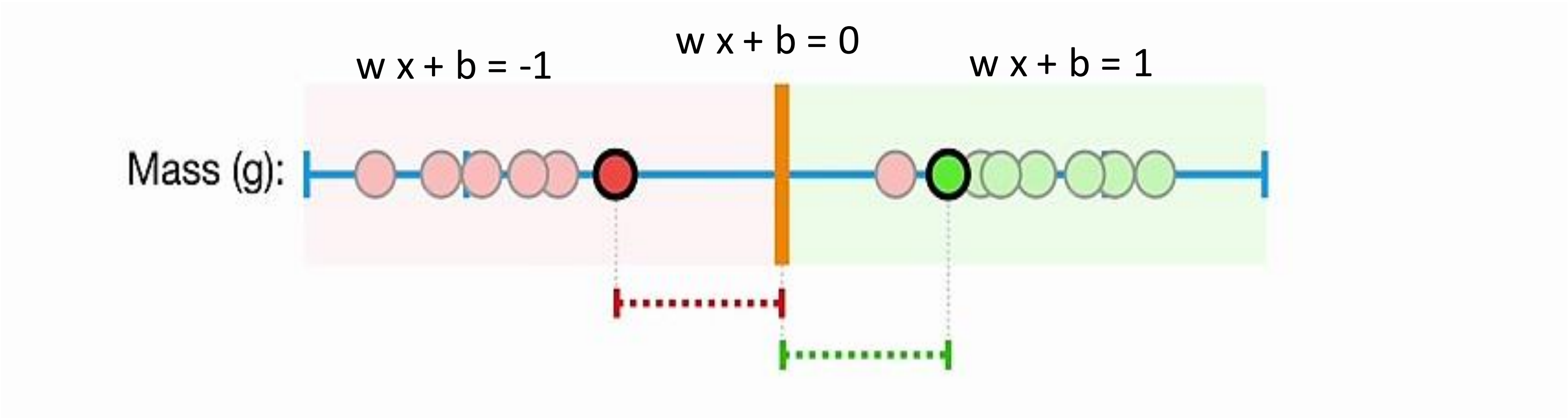
When we use a **Soft Margin** to determine the location of a threshold...



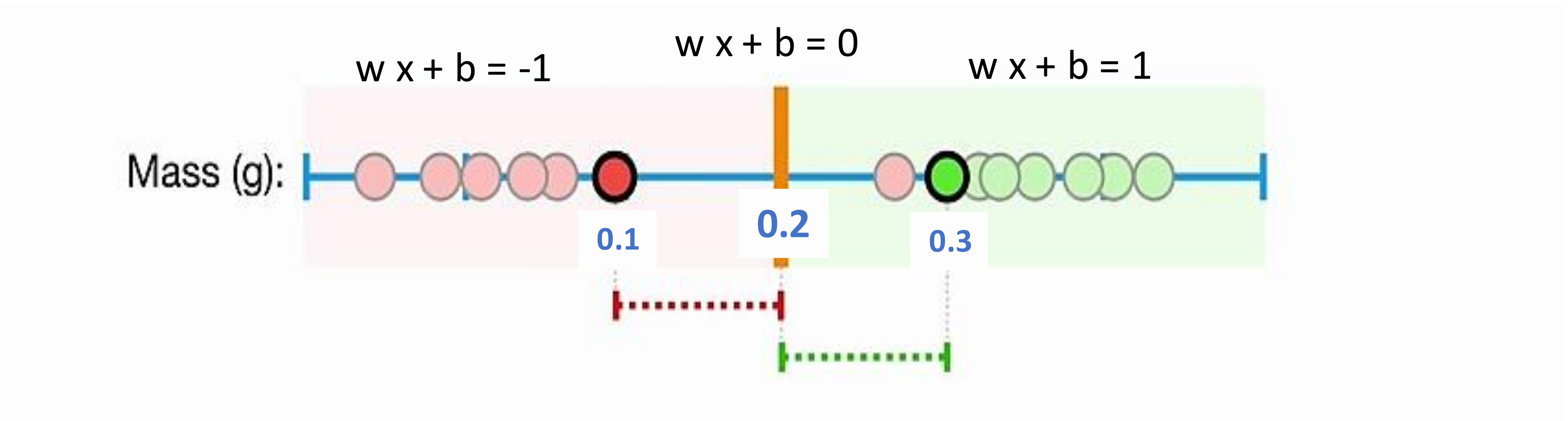
…then we are using a **Soft Margin Classifier** aka a **Support Vector Classifier** to classify observations.



The name **Support Vector Classifier** comes from the fact that the observations on the edge *and within* the **Soft Margin** are called **Support Vectors.**



GREEN if w x + b >= 1
RED if w x + b <= -1



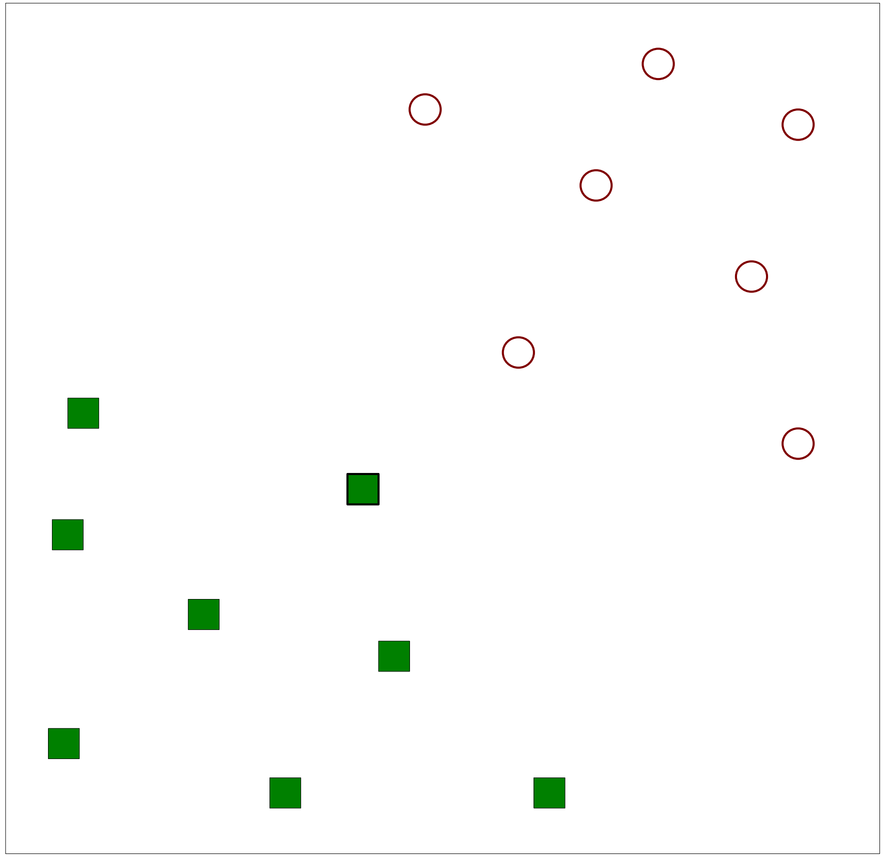
GREEN if 10 x + -2 >= 1

RED if 10 x + -2 <= -1

From One to Two Dimensions

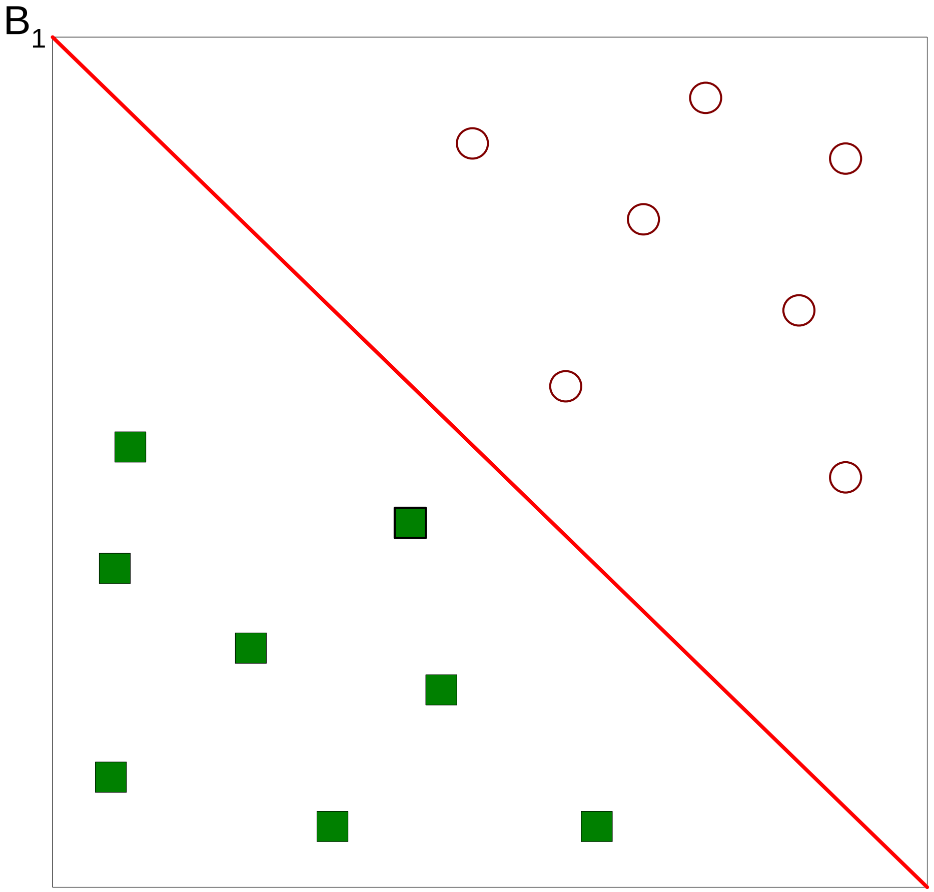
Maximum Margin Hyperplanes

- Find a linear hyperplane (decision boundary) that separates the data.



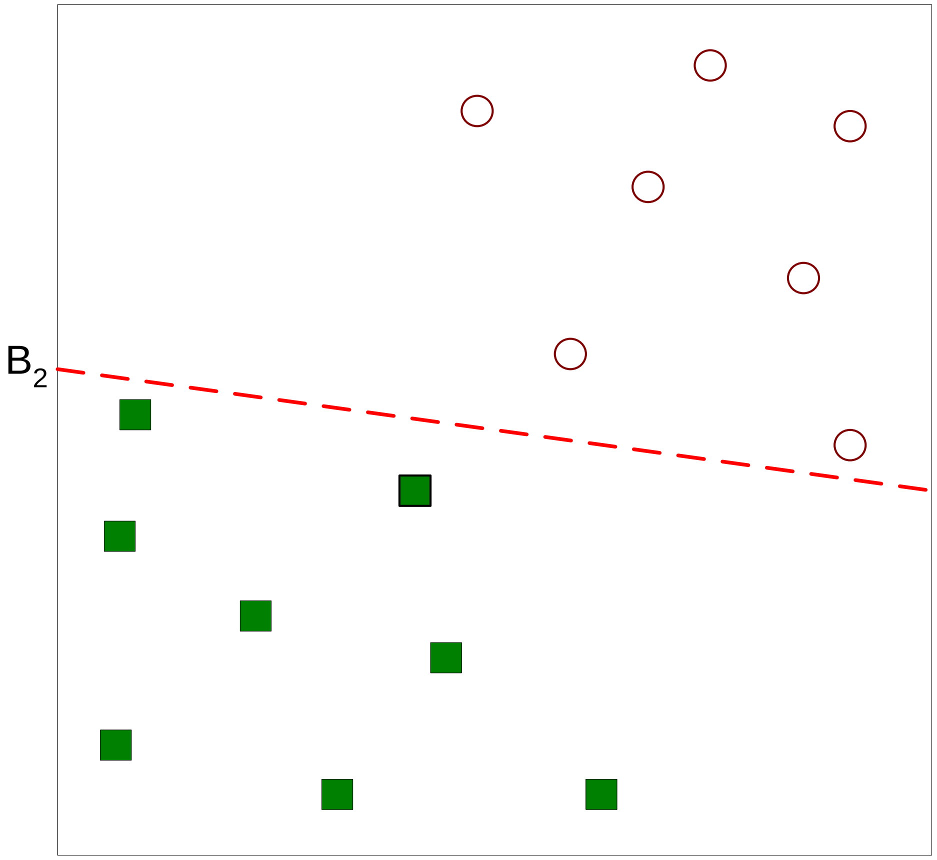
Maximum Margin Hyperplanes

- One possible solution.



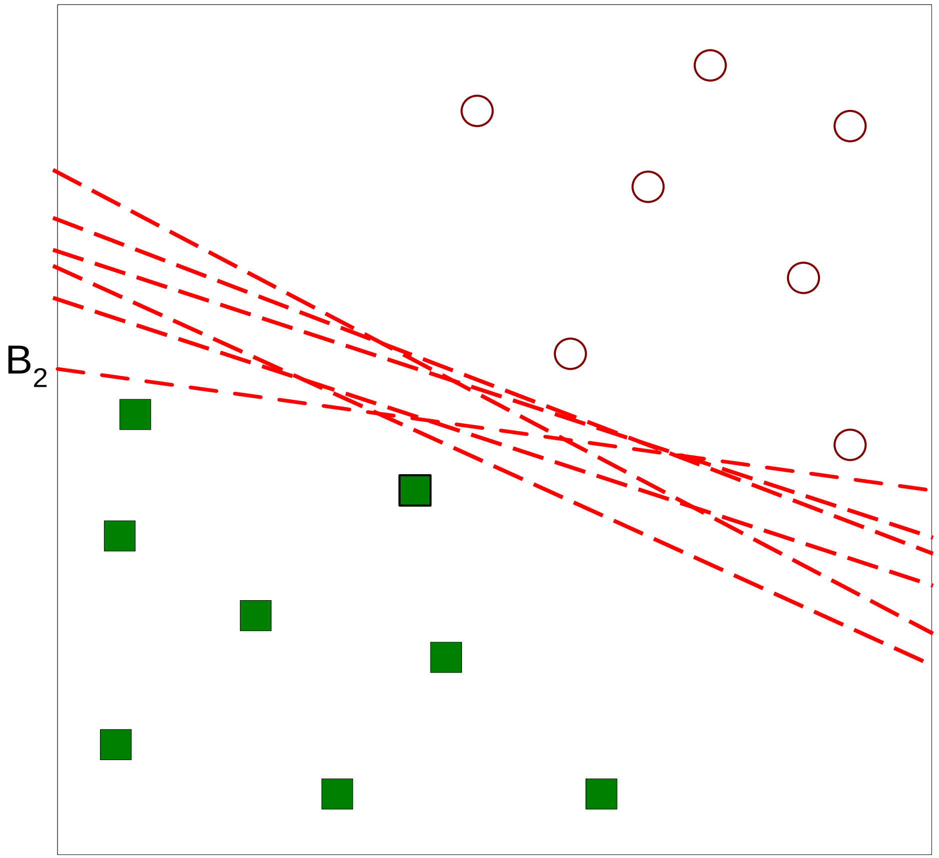
Maximum Margin Hyperplanes

- Another possible solution.



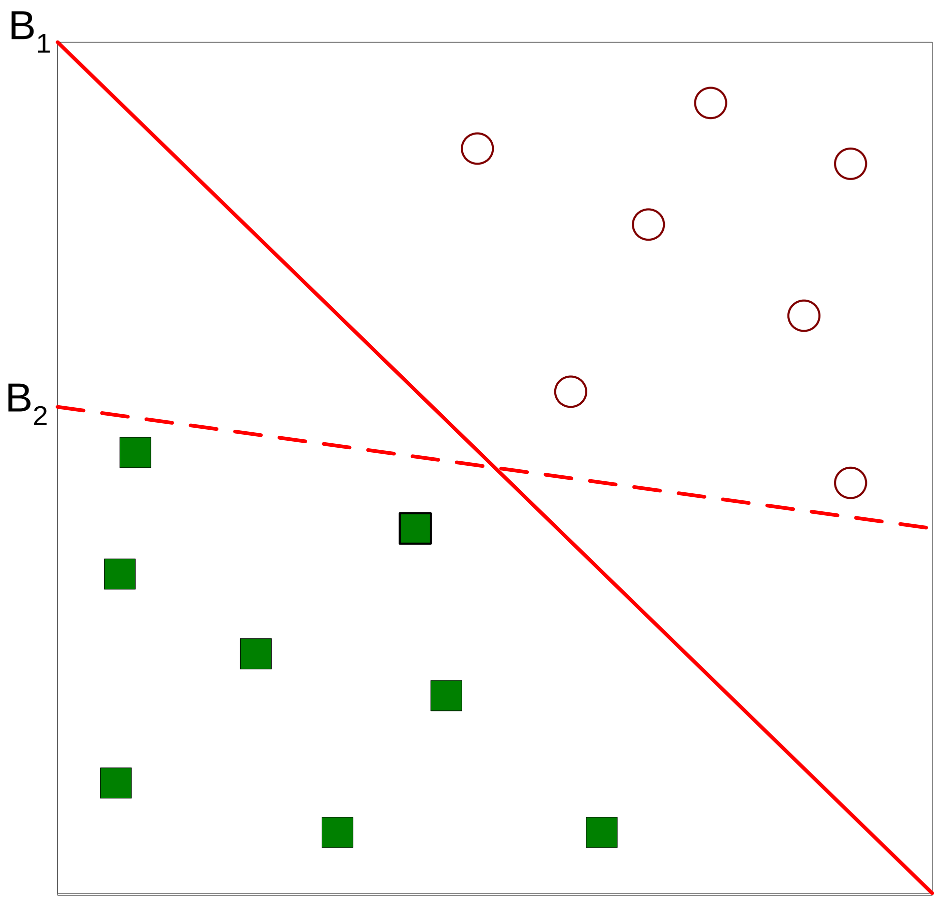
Maximum Margin Hyperplanes

- Other possible solutions.



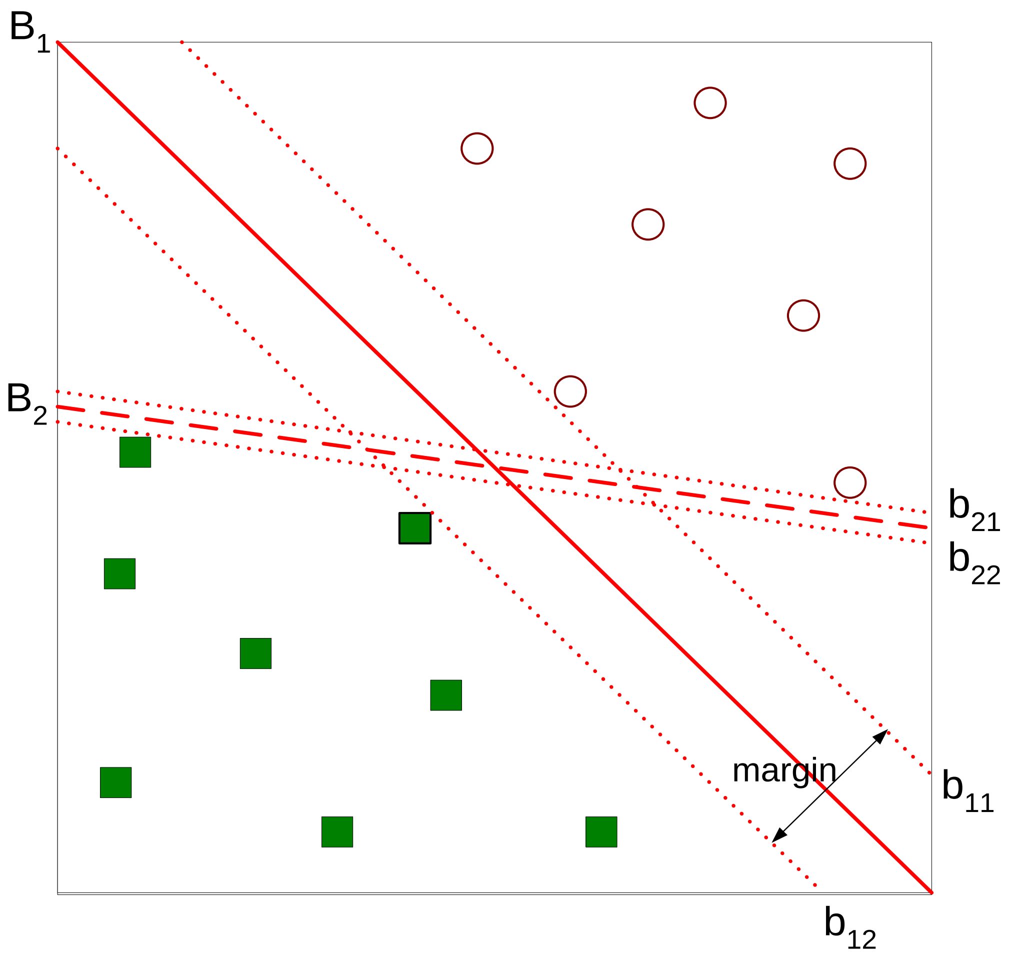
Maximum Margin Hyperplanes

- Let’s focus on B_1 and B_2.
- Which one is better?
- How do you define better?



Maximum Margin Hyperplanes

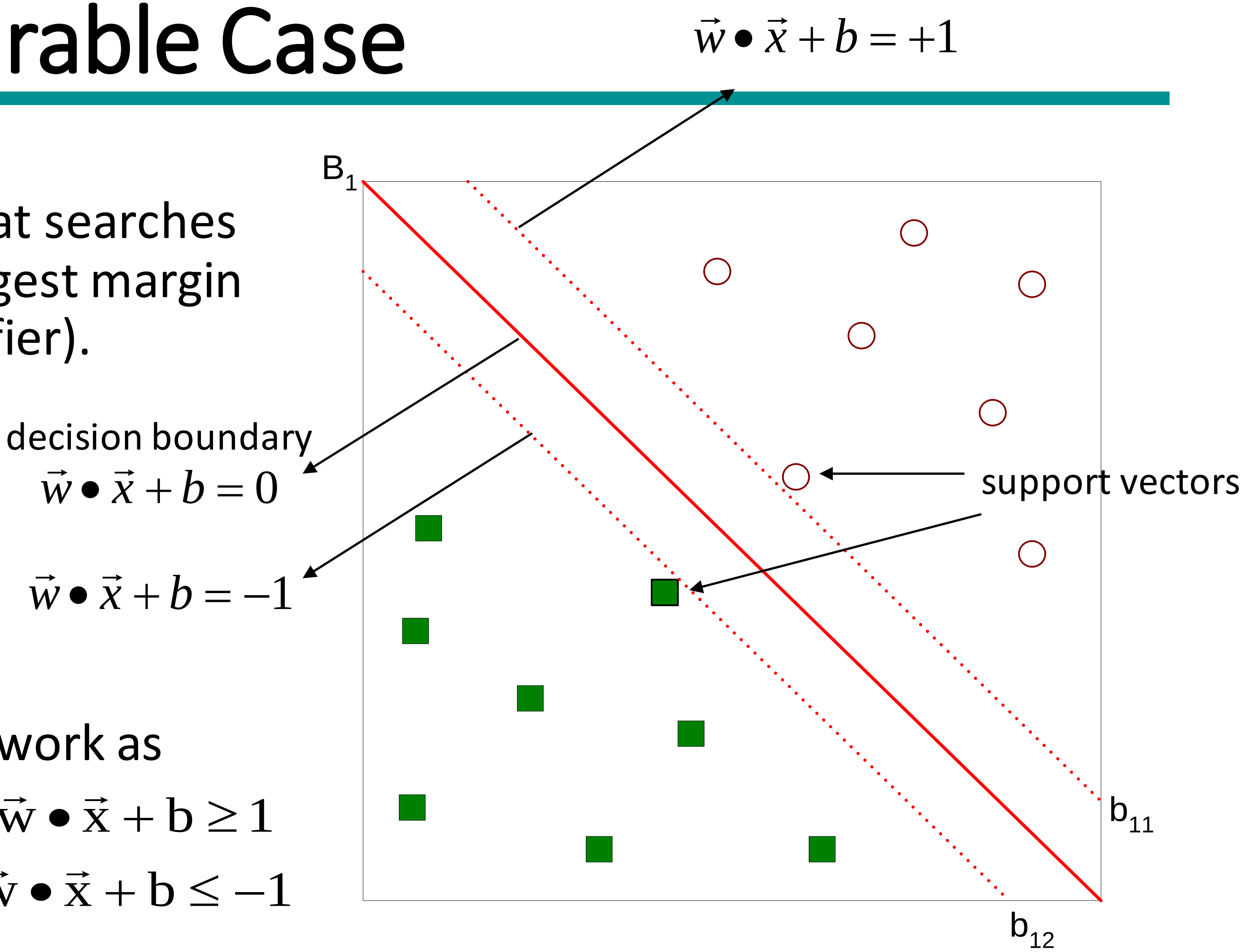
- The best solution is the hyperplane that **maximizes** the **margin**.
- Thus, B_1 is better than B_2.



Linear SVM: Separable Case

- A linear SVM is a classifier that searches for a hyperplane with the largest margin (a.k.a. maximal margin classifier).
- *w* and *b* have to be learned.
- Given *w* and *b* the classifiers work as

$$f(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x} + b \geq 1 \\ -1 & \text{if } \vec{w} \bullet \vec{x} + b \leq -1 \end{cases}$$



Example calculus dot product

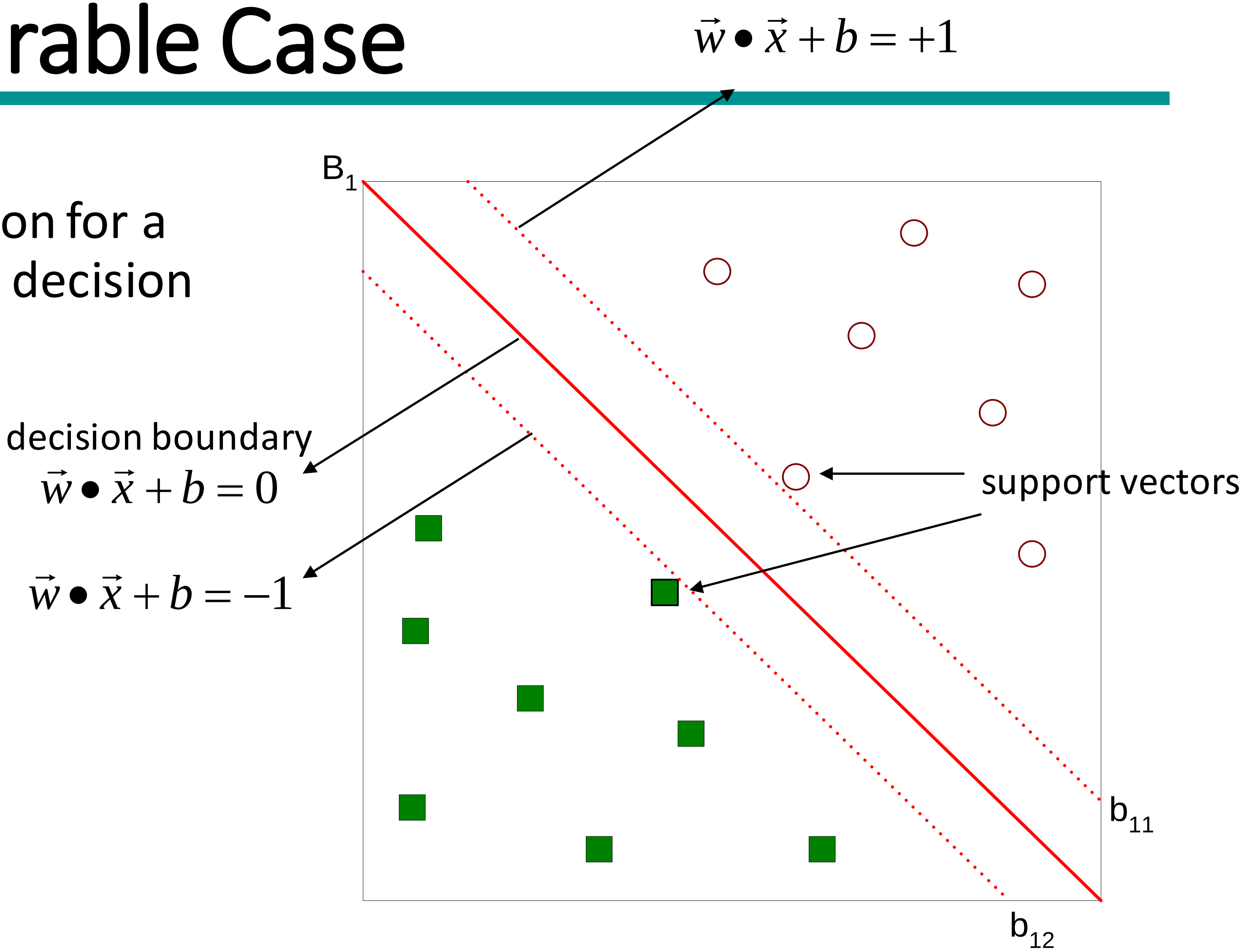
w = [.3 .2]   x = [1 2]   b = -2

w · x + b = .3*1 + .2*2 +(-2) = -1.3

Linear SVM: Separable Case

- What is the distance expression for a point *x* to a line *wx+b= 0* (the decision boundary)?

$$d(\mathbf{x}) = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\|\mathbf{w}\|_2^2}} = \frac{|\mathbf{x}\cdot\mathbf{w}+b|}{\sqrt{\sum_{i=1}^{d} w_i^2}}$$

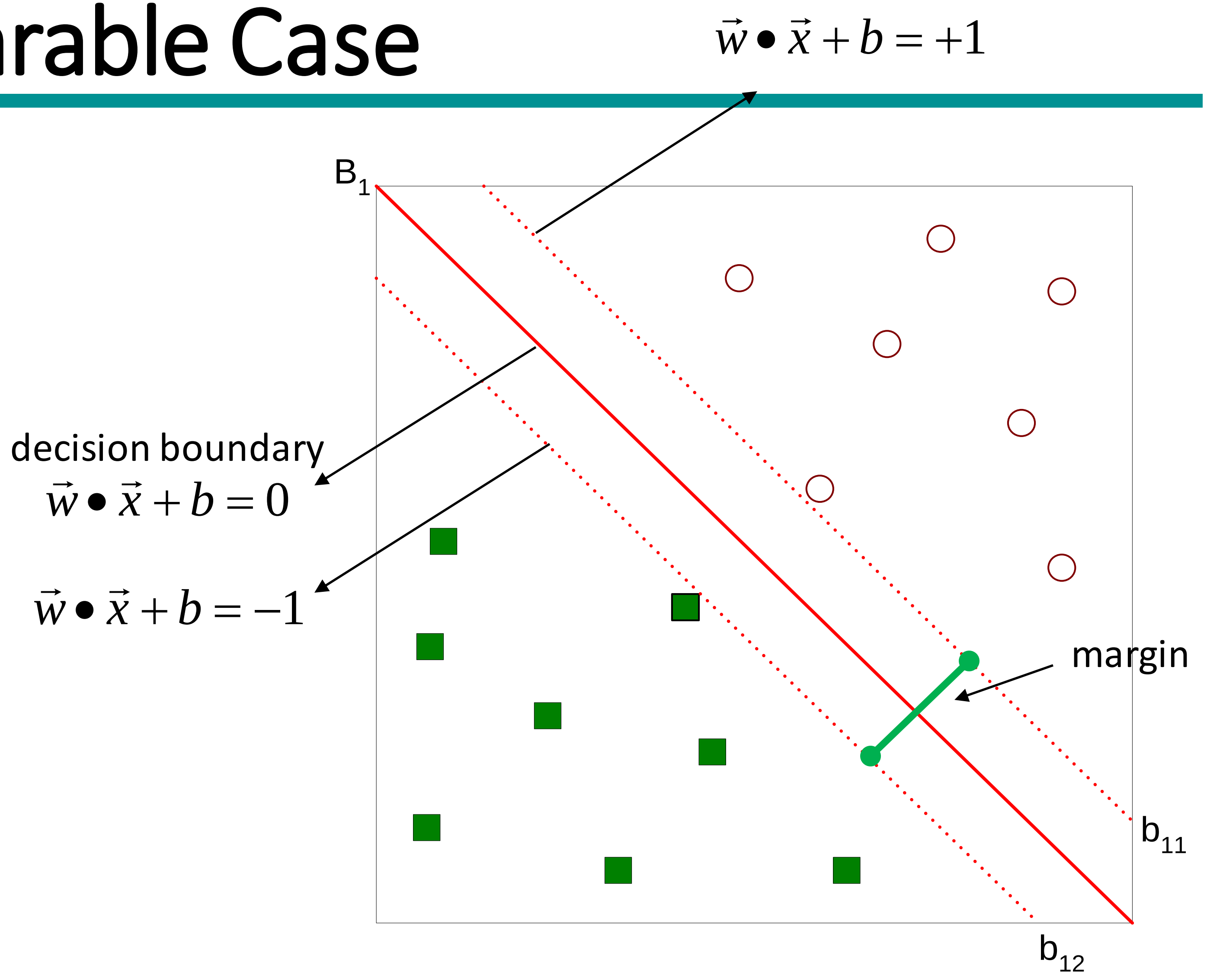


Linear SVM: Separable Case

- The distance between B_1 and b_{11} is $1/||w||$
- The distance between b_{11} and b_{12}, i.e., the margin is

$$\text{Margin} = \frac{2}{\|\vec{w}\|}$$

- In order to ***maximize the margin*** we need to minimize $||w||$



Learning a Linear SVM

- Learning the SVM model is equivalent to determining *w* and *b.*
- How to find *w* and *b?*
- Objective is to ***maximize the margin***.
- Which is equivalent to minimize
- Subject to to the following constraints
- This is a constrained optimization problem that can be solved using the *Lagrange* multiplier method.
- Introduce Lagrange multiplier λ (or α)

$$\text{Margin} = \frac{2}{\|\vec{w}\|}$$

$$L(\vec{w}) = \frac{\|\vec{w}\|^2}{2}$$

$$y_i = \begin{cases} 1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_\mathrm{i} + \mathrm{b} \geq 1 \\ -1 & \text{if } \vec{\mathrm{w}} \bullet \vec{\mathrm{x}}_\mathrm{i} + \mathrm{b} \leq -1 \end{cases}$$

=

$$y_i(\mathbf{w} \bullet \mathbf{x}_i + b) \geq 1, \quad i = 1,2,\ldots,N$$

Constrained Optimization Problem

$\text{Minimize} \| \mathbf{w} \| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method : maximize $\inf_{\mathbf{w}} L(\mathbf{w}, b, \alpha)$, where

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_i \alpha_i \left[\left(y_i (\mathbf{x}_i \cdot \mathbf{w}) + b \right) - 1 \right]$$

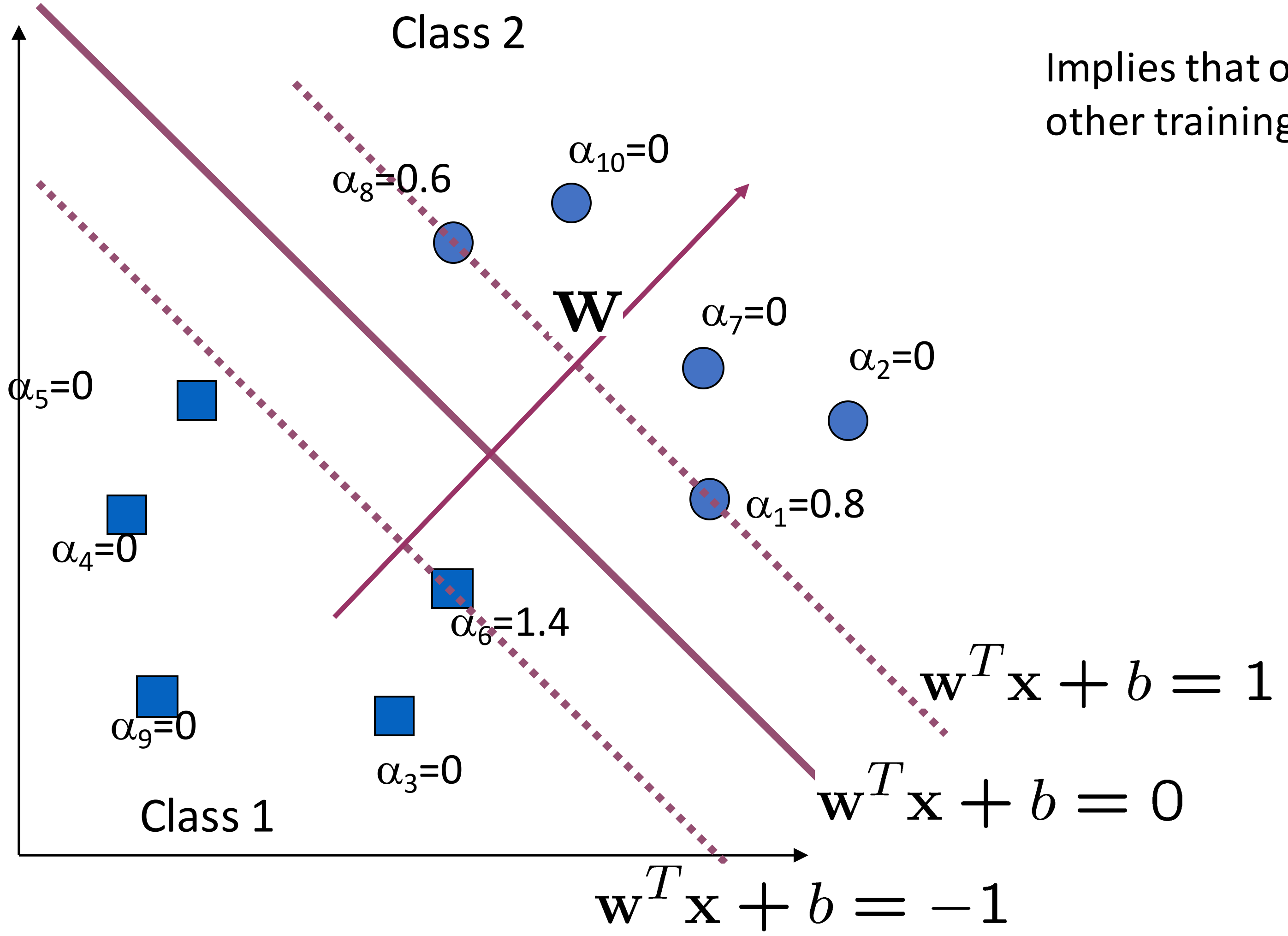
At the extremum, the partial derivative of L with respect both $\mathbf{w}$ and b must be 0. Taking the derivatives, setting them to 0, substituting back into L, and simplifying yields:

$$\text{Maximize} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$$

$$\text{subject to} \sum_i y_i \alpha_i = 0 \text{ and } \alpha_i \geq 0$$

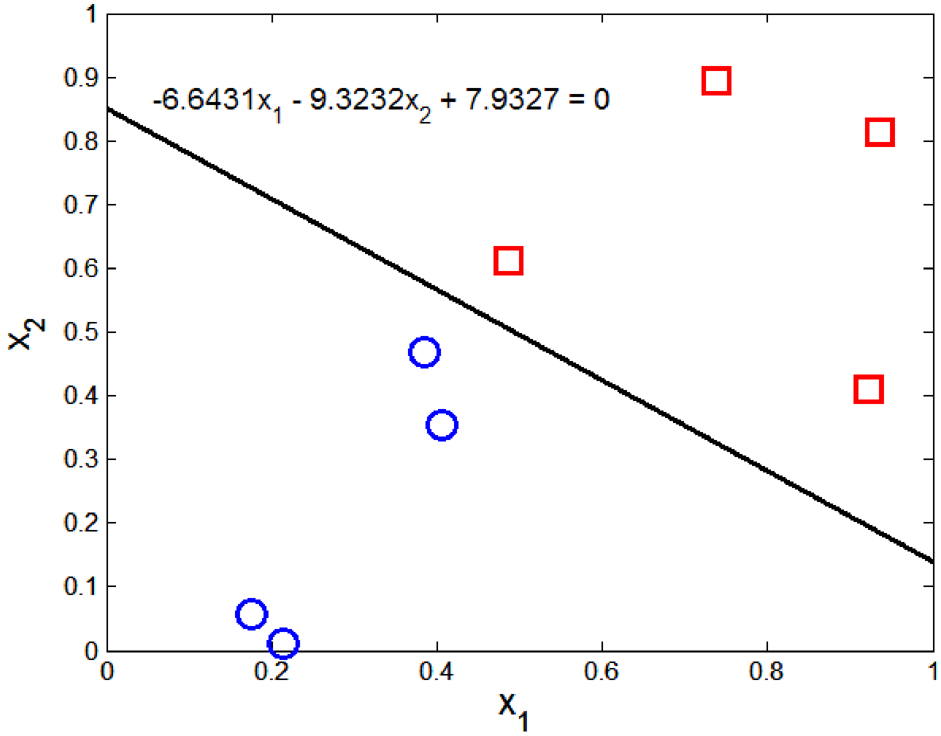
Lagrange multiplier method is a technique for finding a maximum or minimum of a function F subject to a constraint.

A Geometrical Interpretation



Implies that only support vectors matter; other training examples are ignorable.

Example of Linear SVM



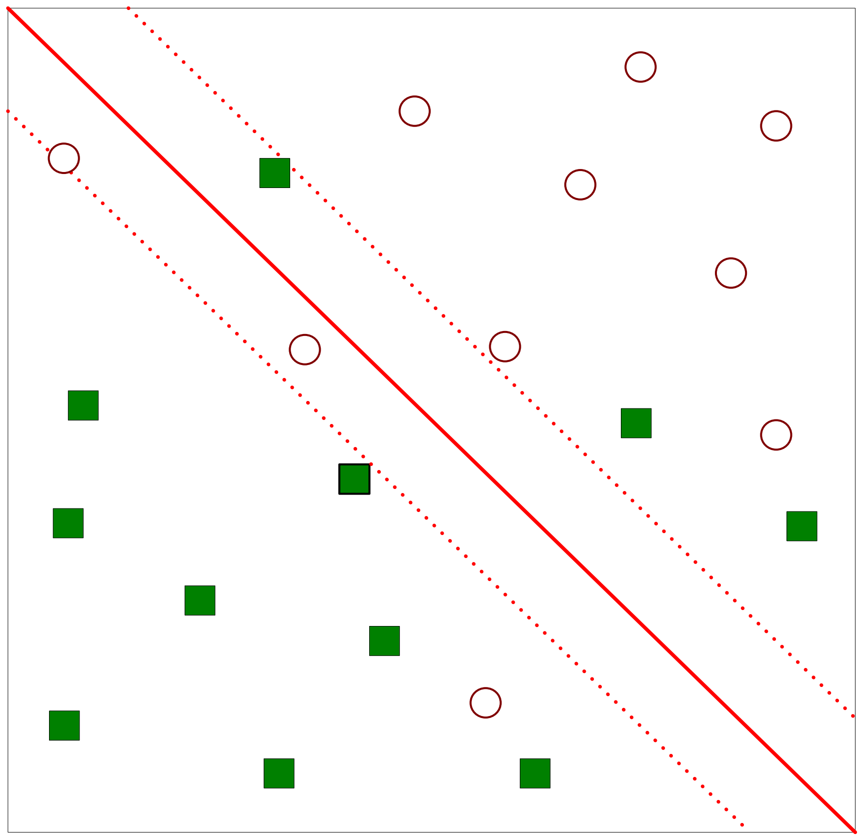
Support vectors

x1	x2	y	λ
0.3858	0.4687	1	65.5261
0.4871	0.611	-1	65.5261
0.9218	0.4103	-1	0
0.7382	0.8936	-1	0
0.1763	0.0579	1	0
0.4057	0.3529	1	0
0.9355	0.8132	-1	0
0.2146	0.0099	1	0

$\lambda = \alpha$

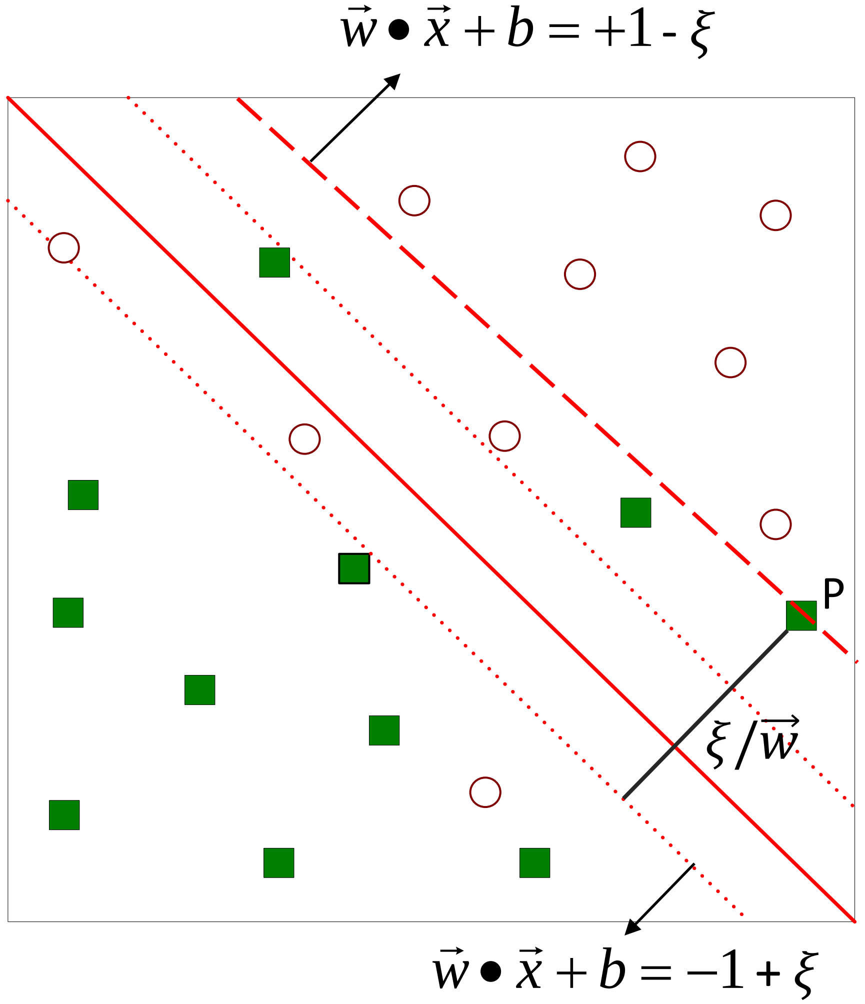
Linear SVM: Non-separable Case

- What if the problem is not linearly separable?
- We must allow for errors in our solution.



Slack Variables

- The inequality constraints must be relaxed to accommodate the nonlinearly separable data.
- This is done introducing slack variables ξ (xi) into the constrains of the optimization problem.
- ξ provides an estimate of the error of the decision boundary on the misclassified training examples.



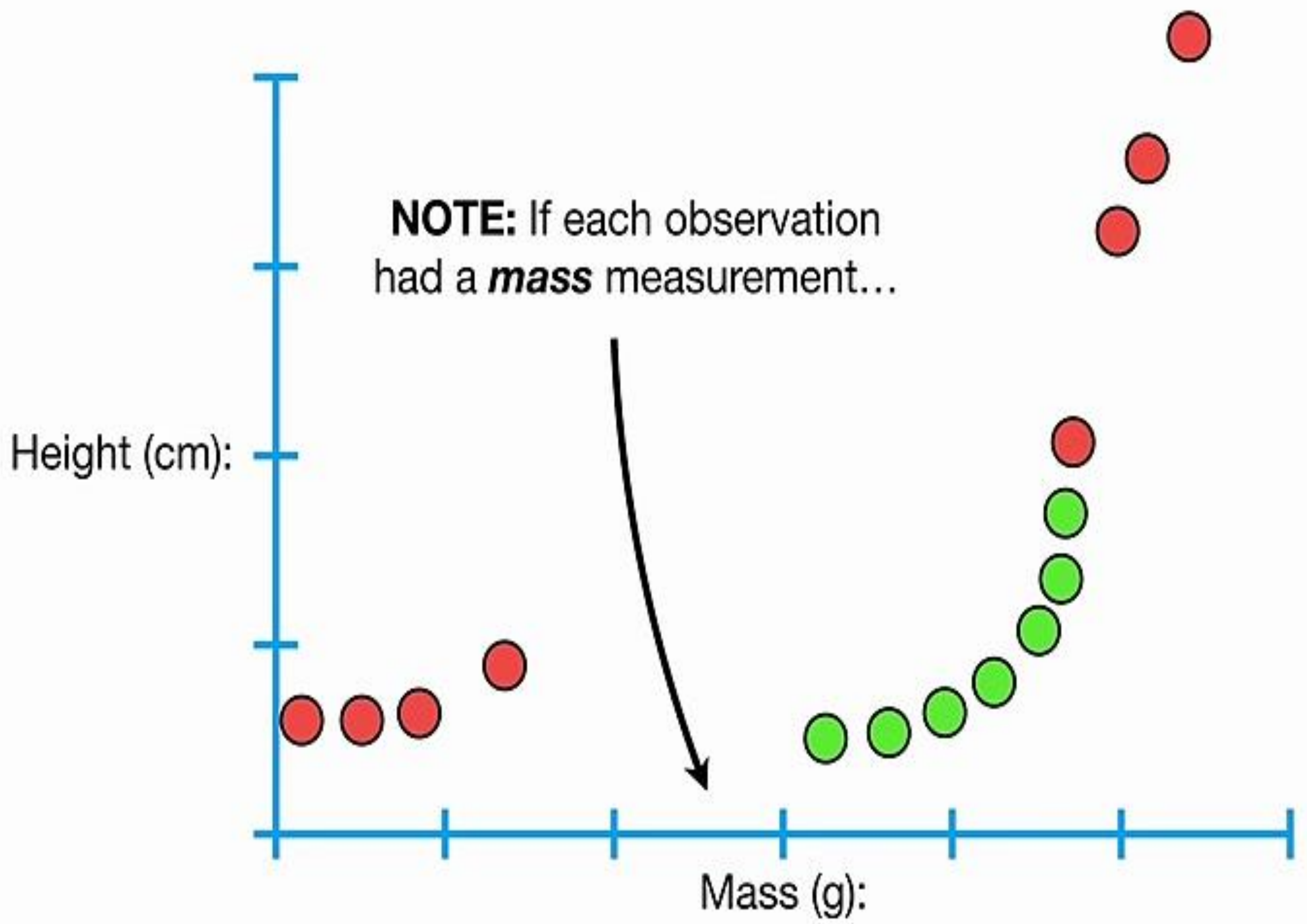
Learning a Non-separable Linear SVM

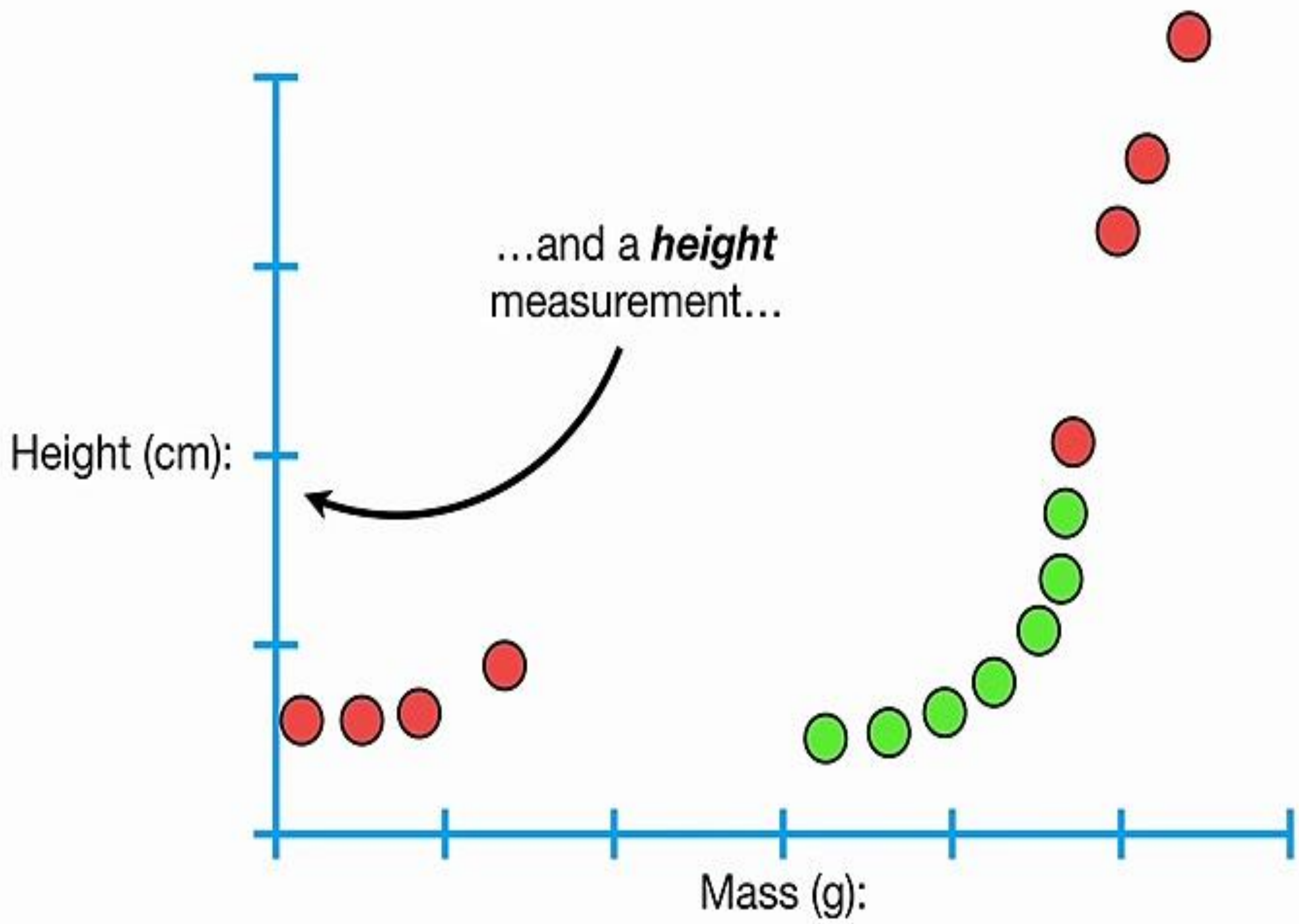
- Objective is to minimize
- Subject to to the constraints
- where *C* and *k* are user-specified parameters representing the penalty of misclassifying the training instances
- Lagrangian multipliers are constrained to $0 \le \lambda \le C$.

$$L(w) = \frac{\| \vec{w} \|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k \right)$$

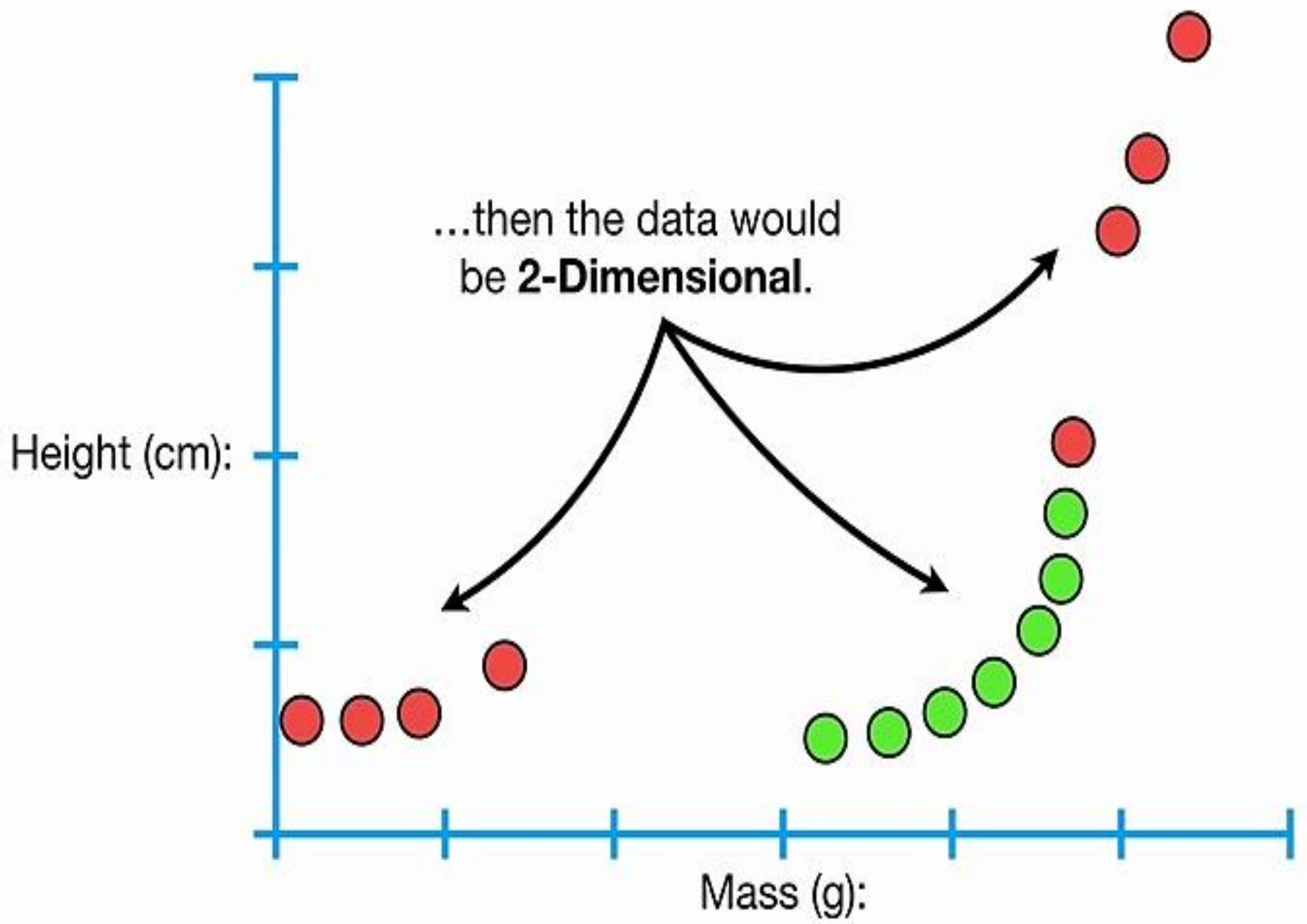
$$y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \ge 1 - \xi_i \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \le -1 + \xi_i \end{cases}$$

https://scikit-learn.org/stable/auto_examples/svm/plot_linearsvc_support_vectors.html#sphx-glr-auto-examples-svm-plot-linearsvc-support-vectors-py

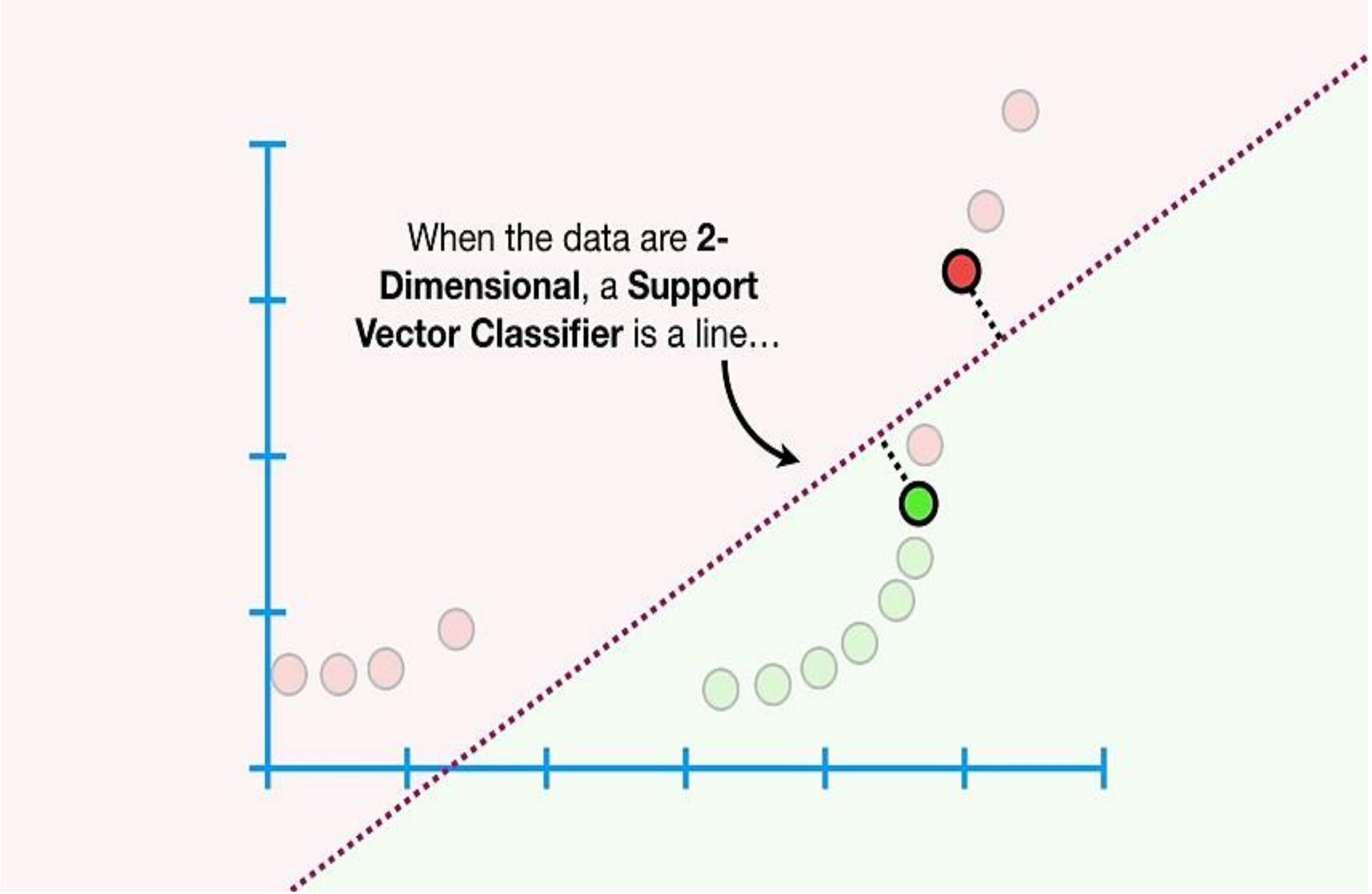

NOTE: If each observation had a mass measurement…
Height (cm):
Mass (g):

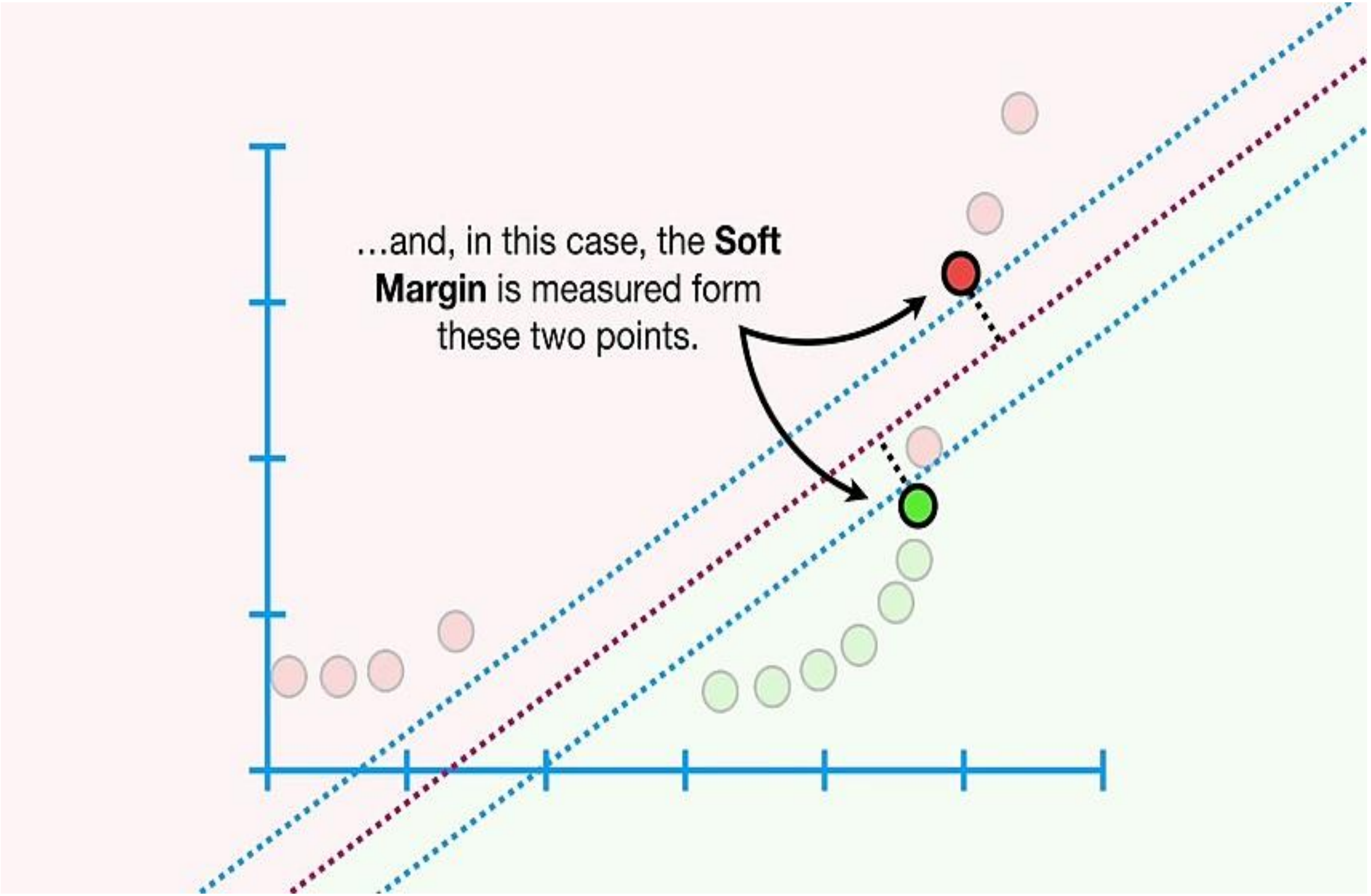


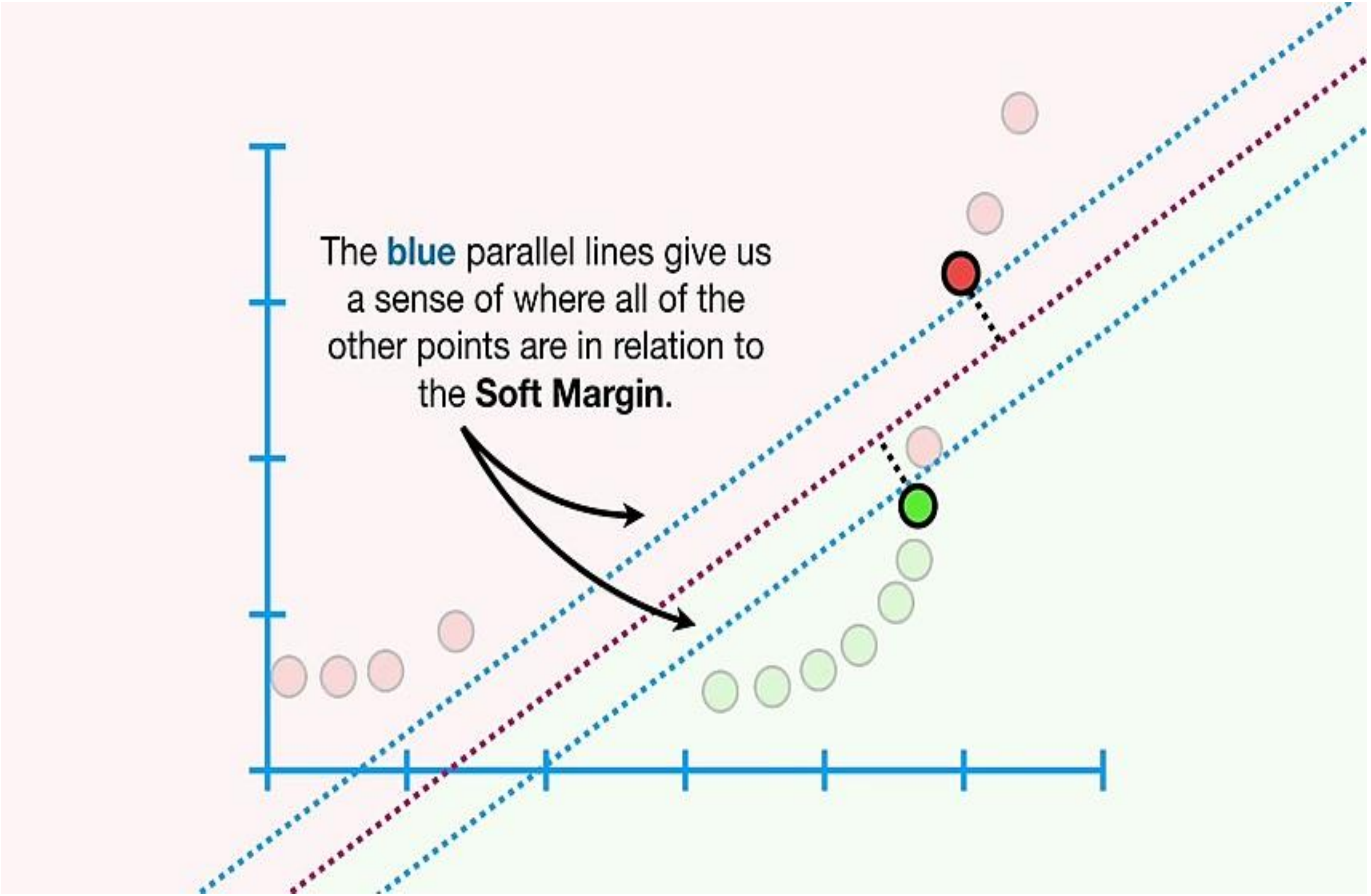
...and a **height** measurement...
Height (cm):
Mass (g):

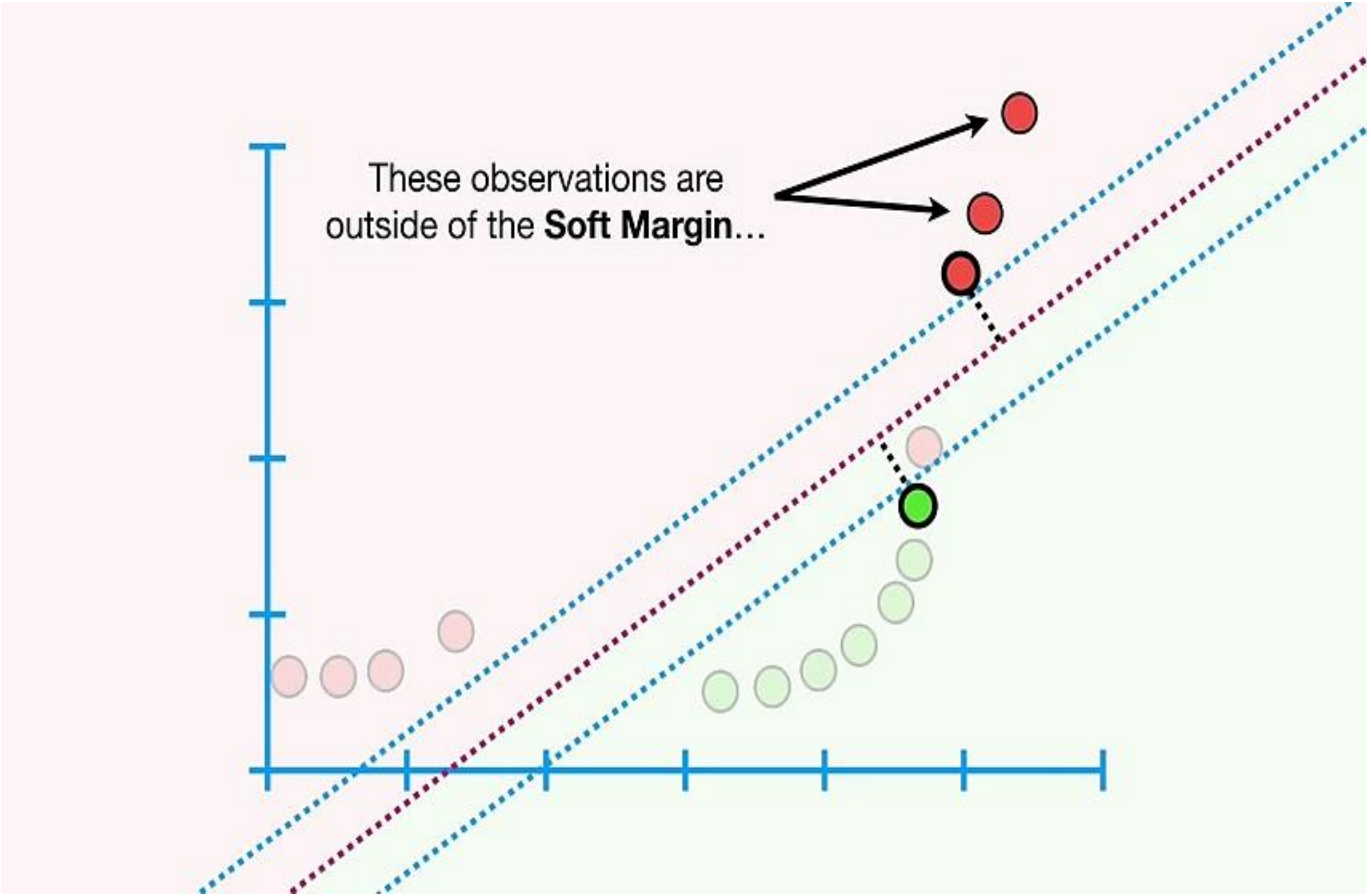


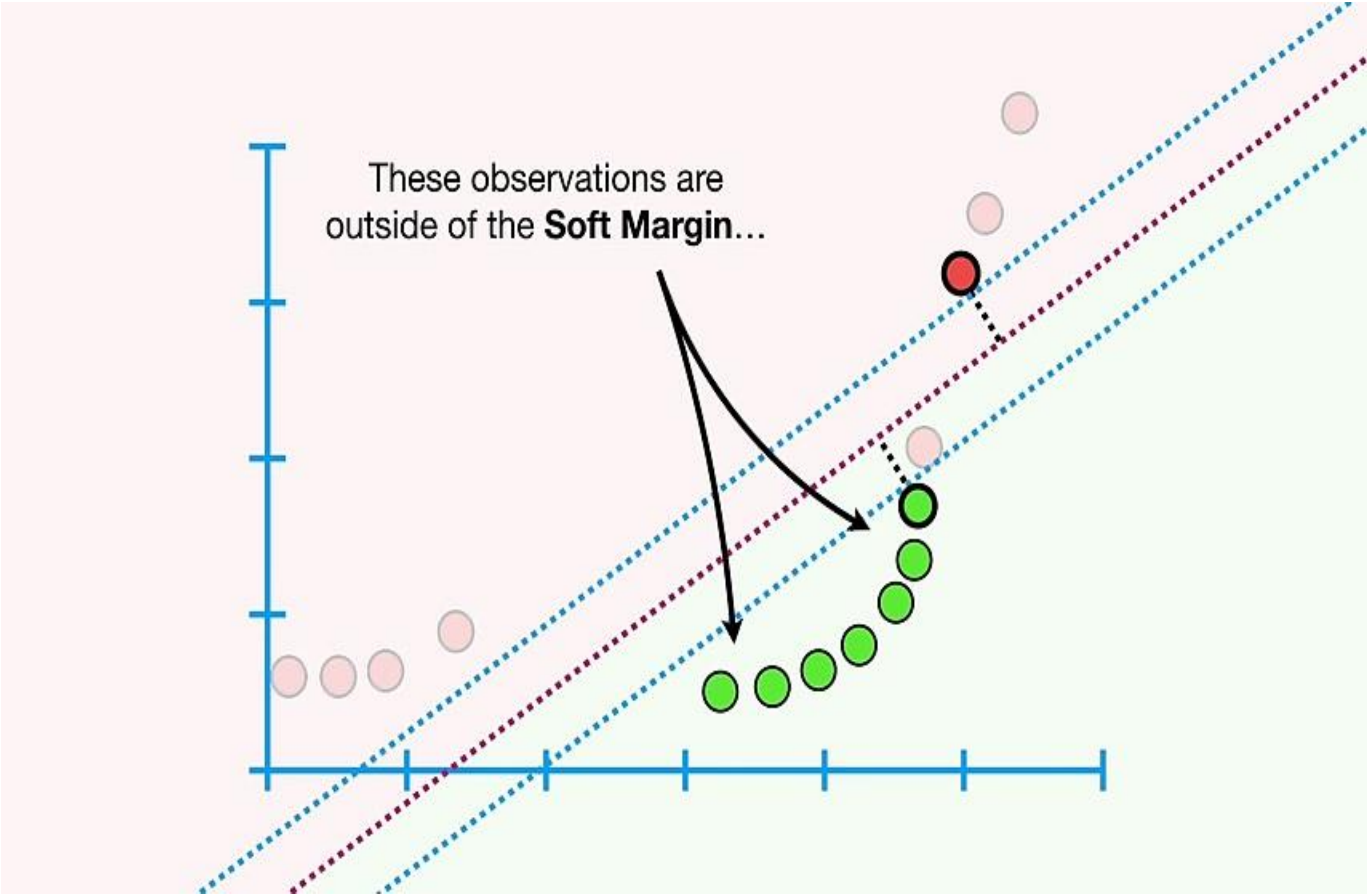
...then the data would
be **2-Dimensional**.
Height (cm):
Mass (g):

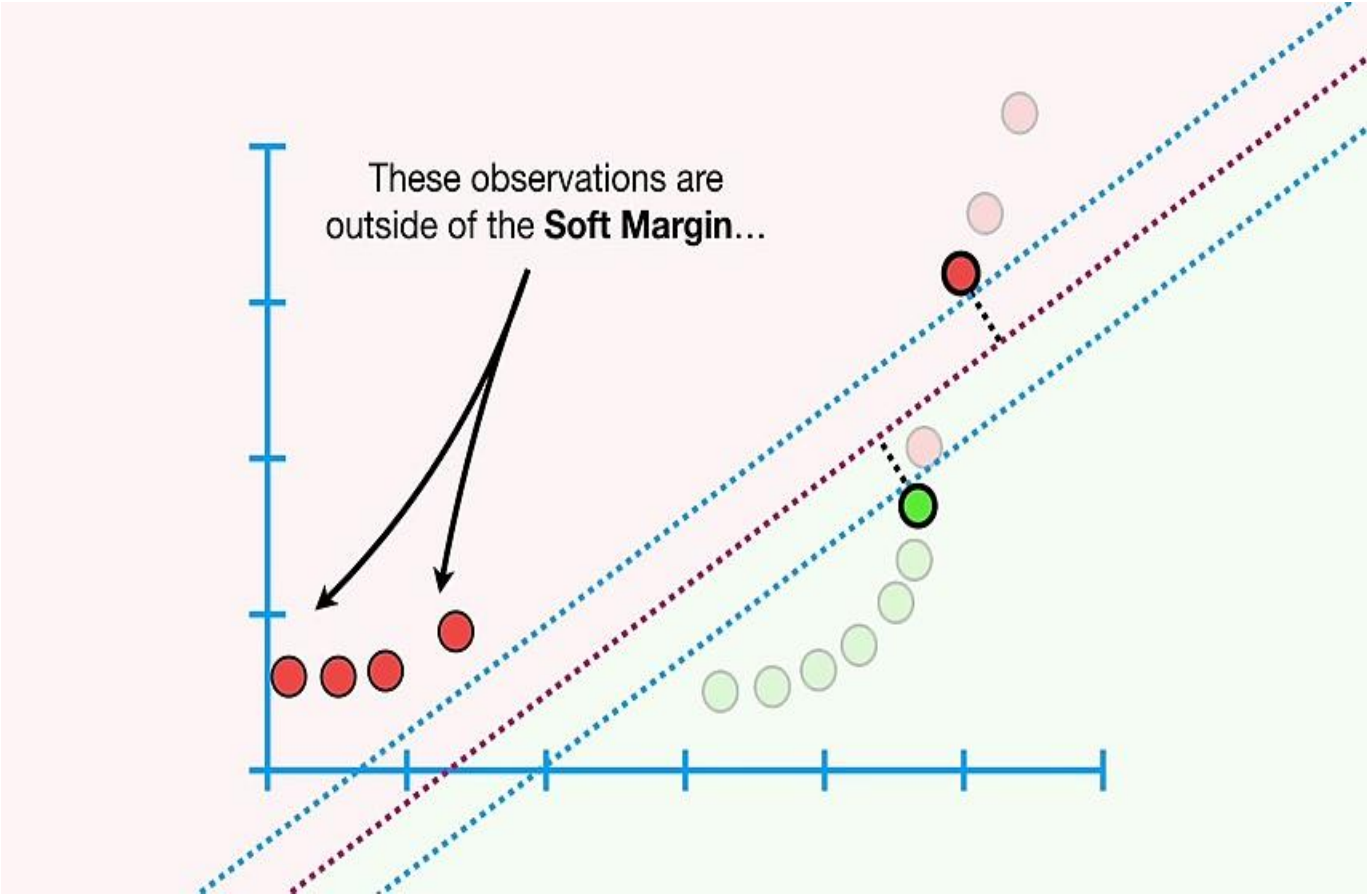

When the data are **2-Dimensional**, a **Support Vector Classifier** is a line…

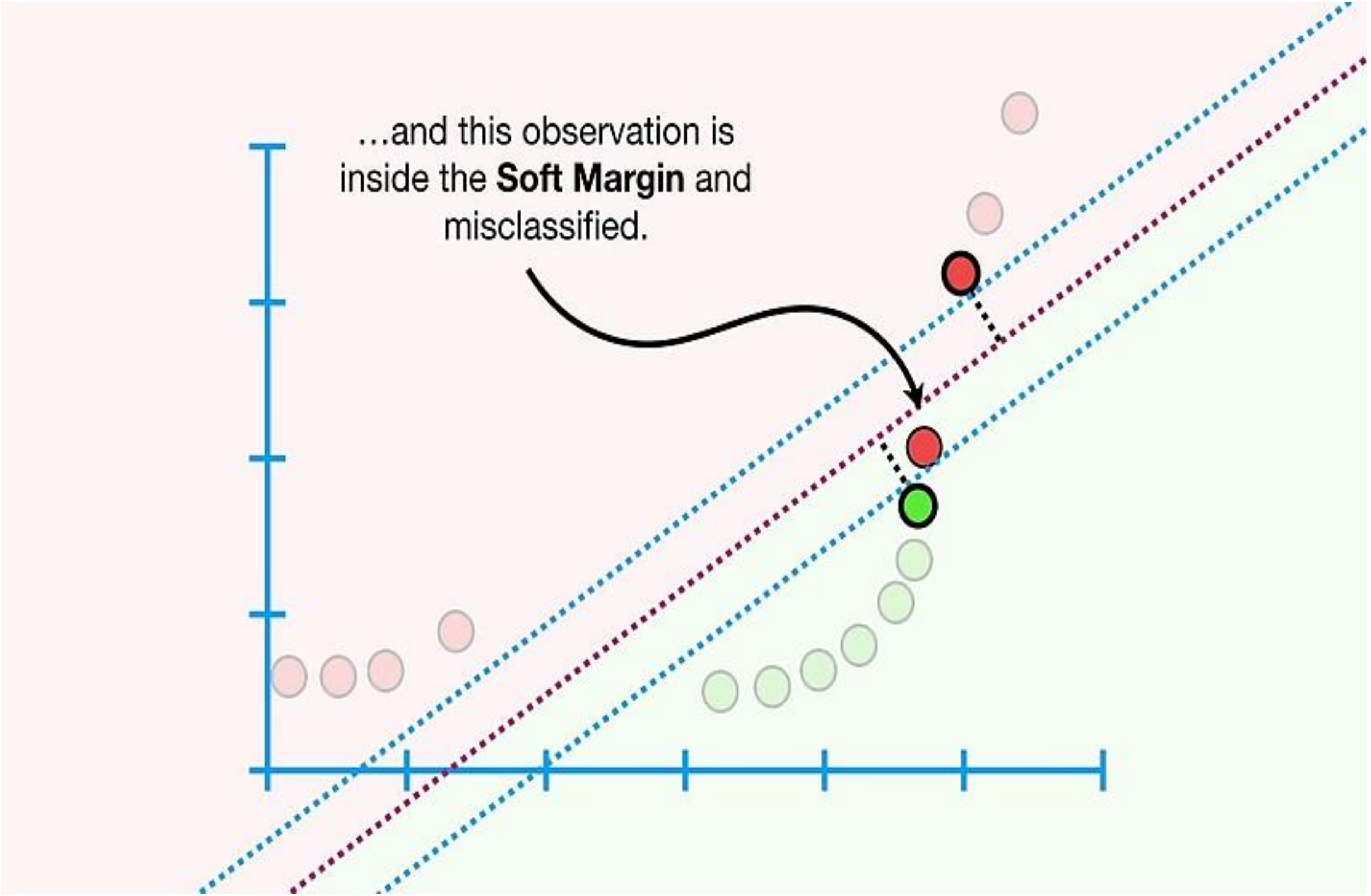

...and, in this case, the **Soft Margin** is measured form these two points.

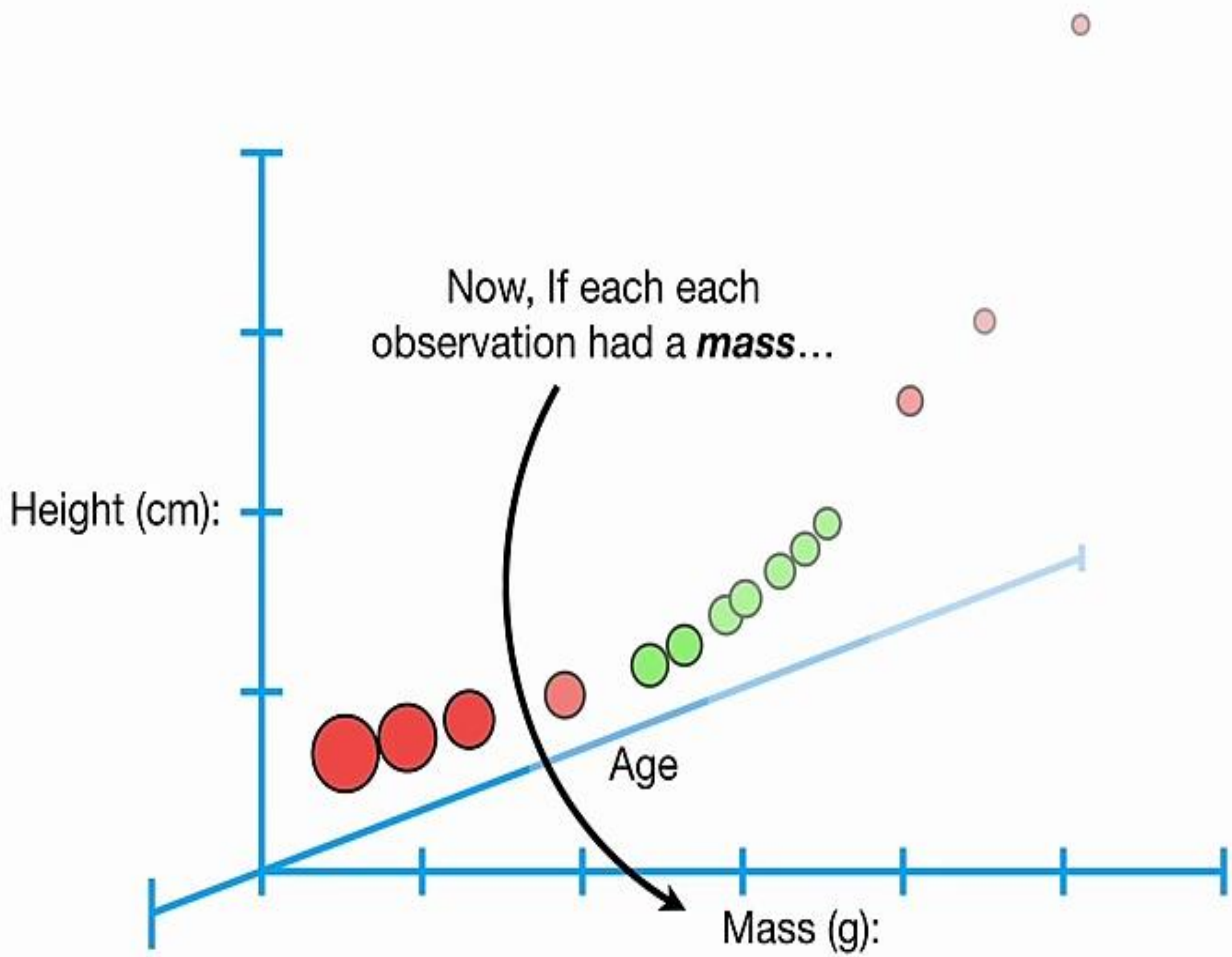

The blue parallel lines give us
a sense of where all of the
other points are in relation to
the Soft Margin.

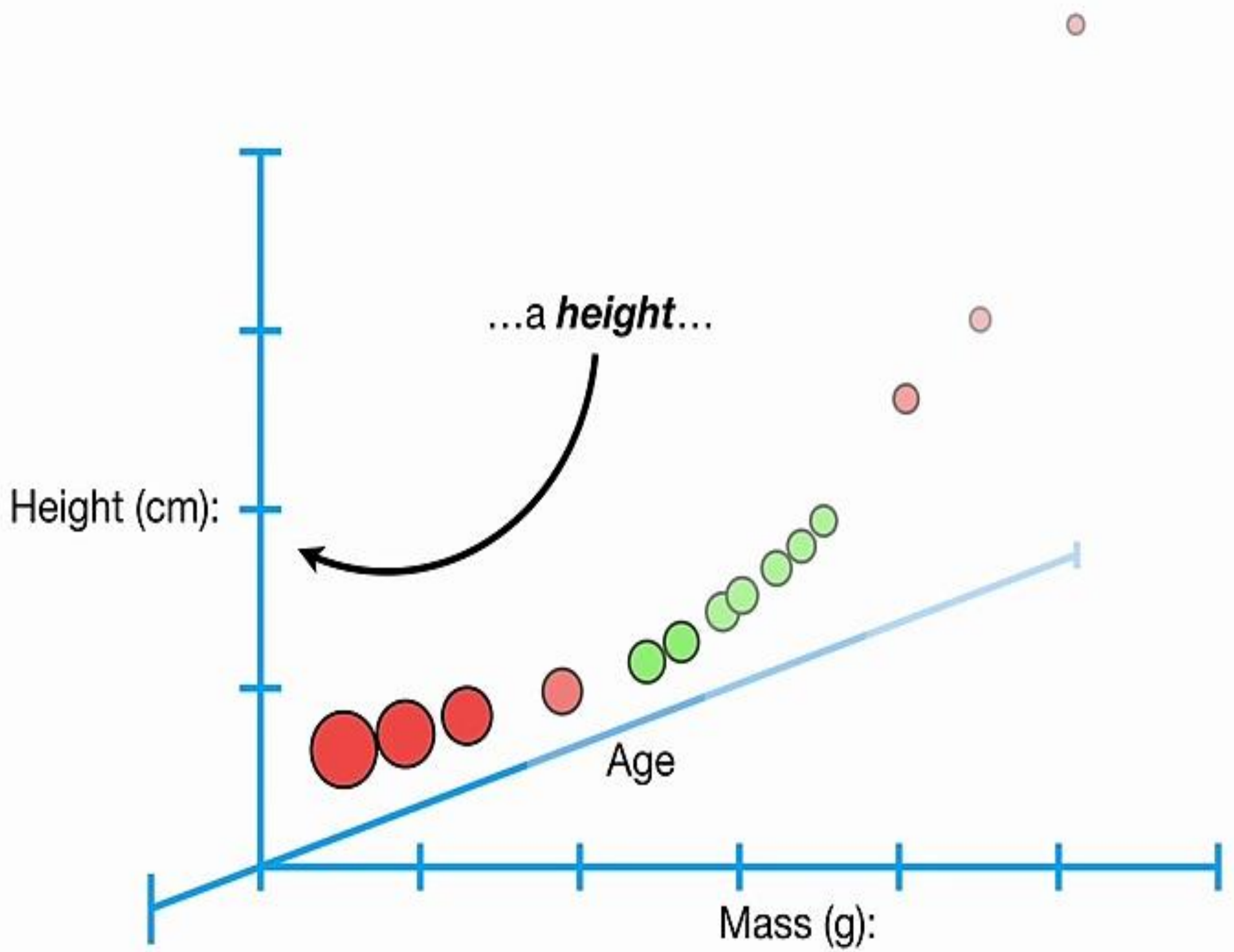

These observations are outside of the **Soft Margin**...


These observations are
outside of the **Soft Margin**...

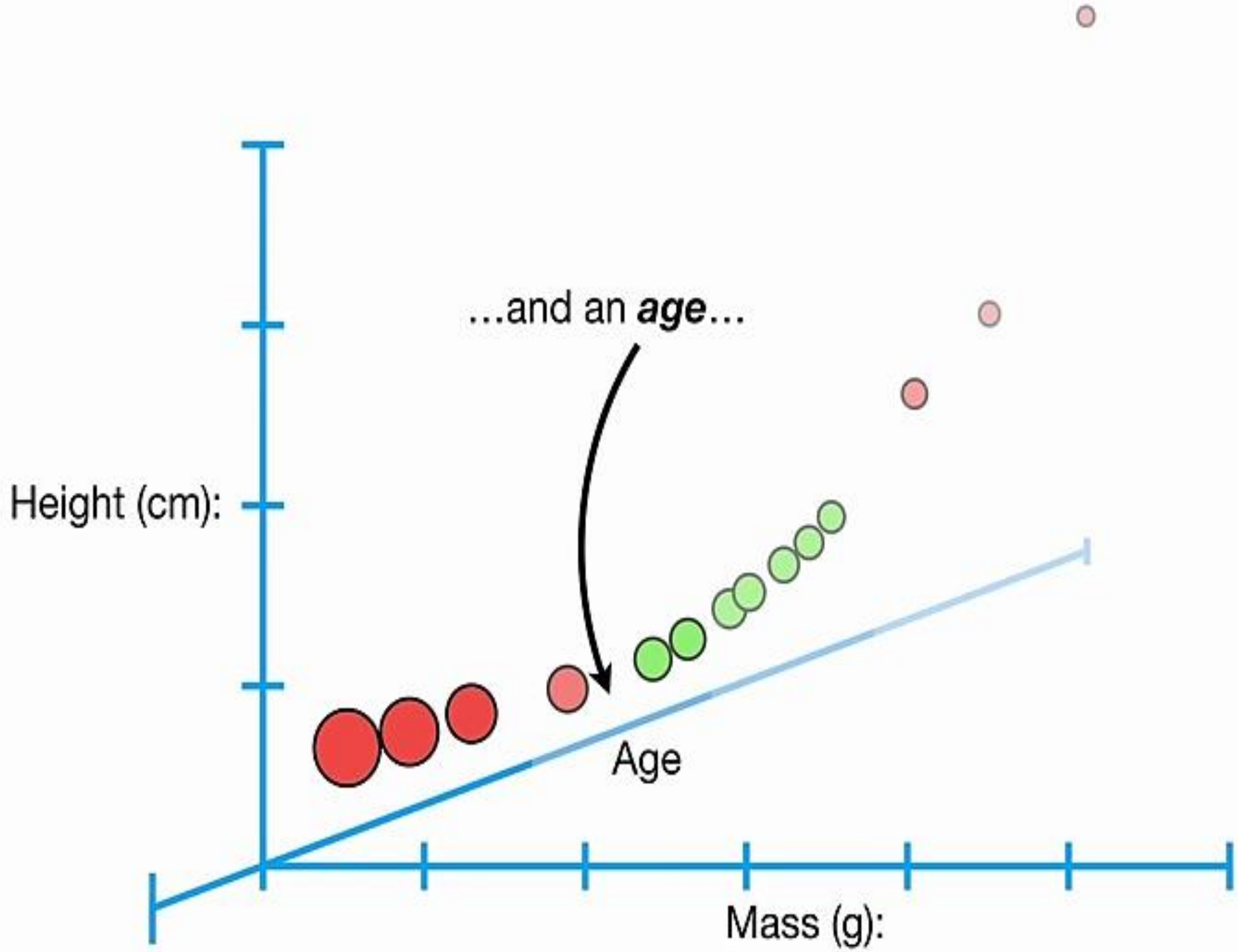

These observations are
outside of the **Soft Margin**...


...and this observation is inside the **Soft Margin** and misclassified.

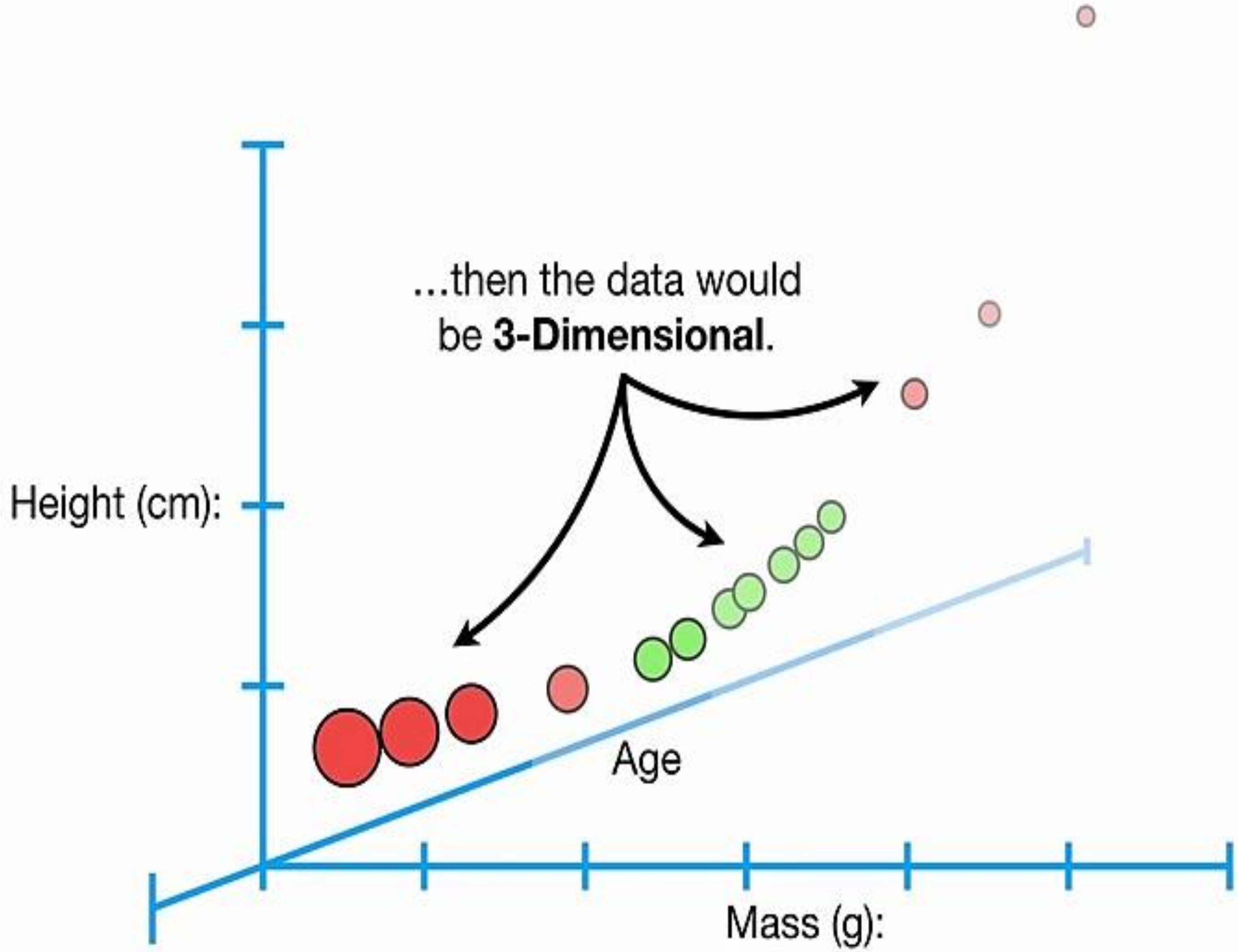

Now, If each each
observation had a ***mass***…
Height (cm):
Age
Mass (g):



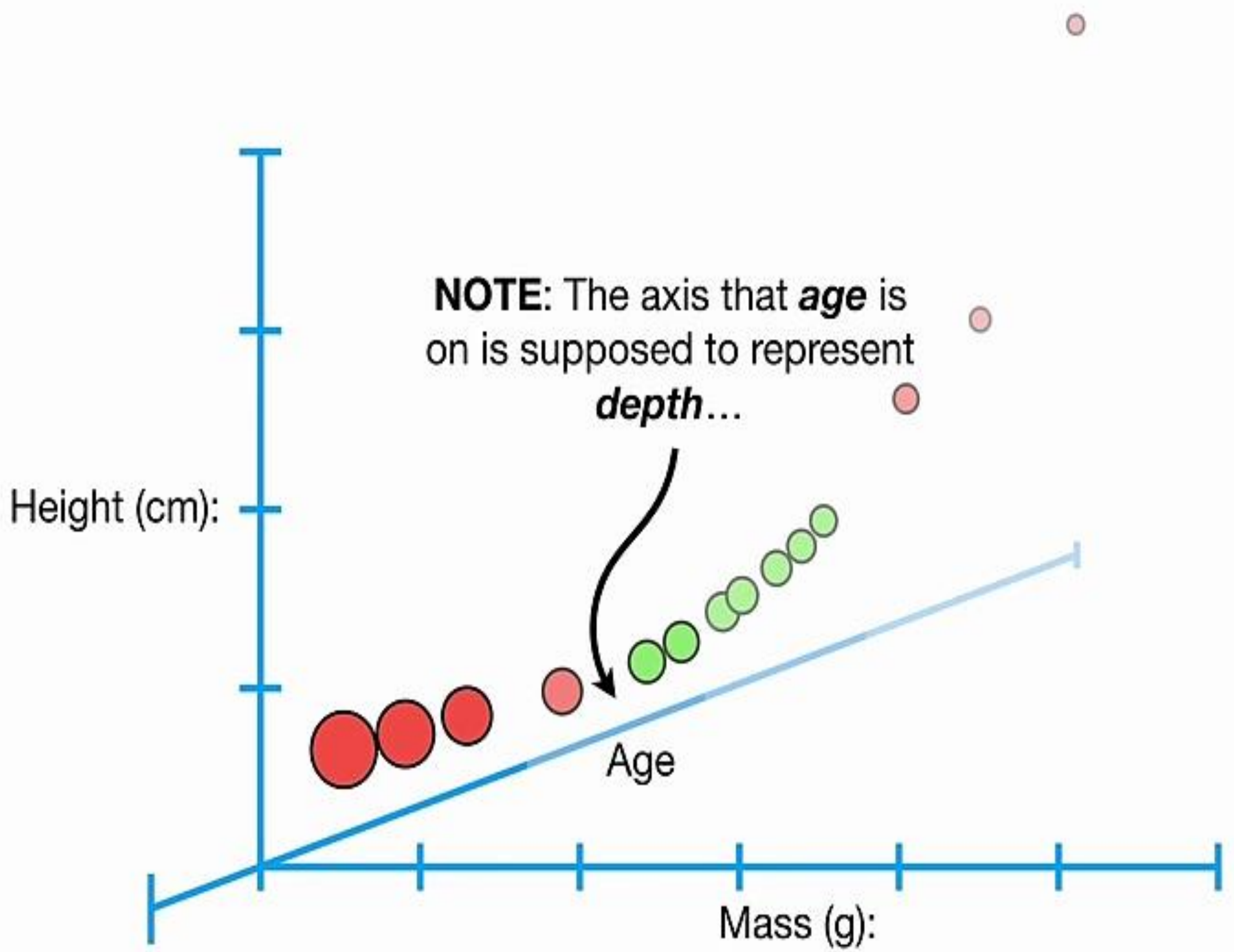
...a ***height***...
Height (cm):
Age
Mass (g):

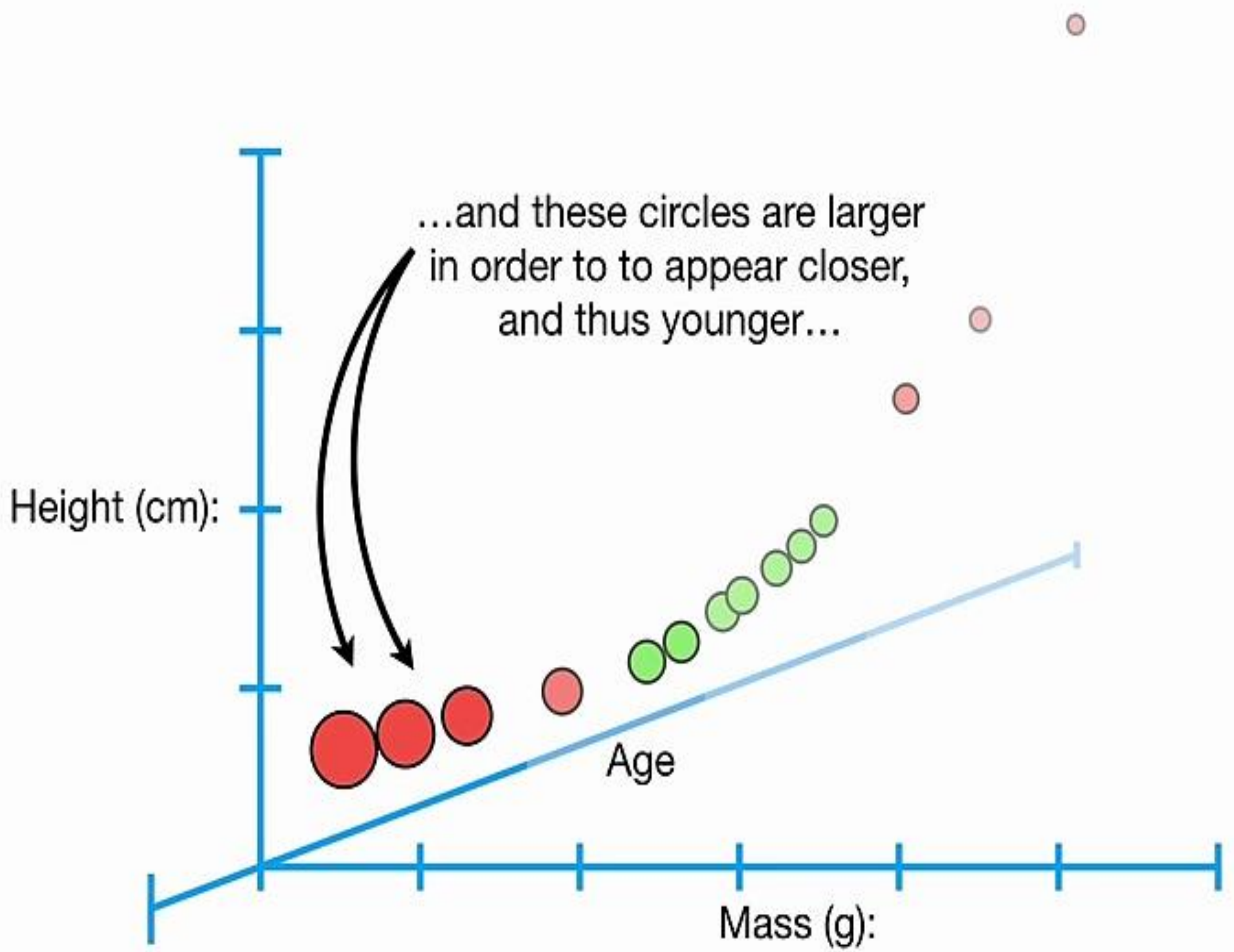


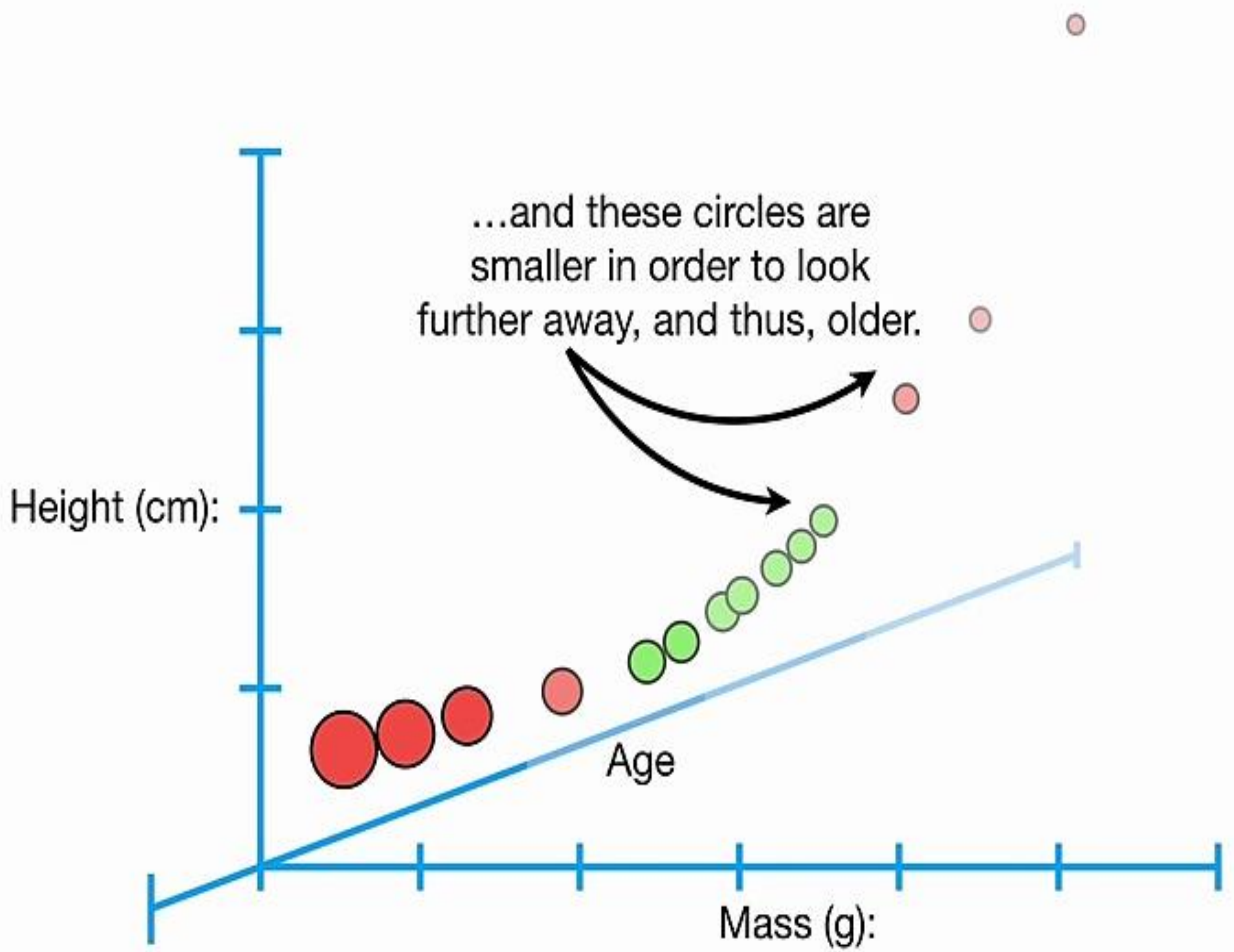
...and an ***age***...
Height (cm):
Age
Mass (g):

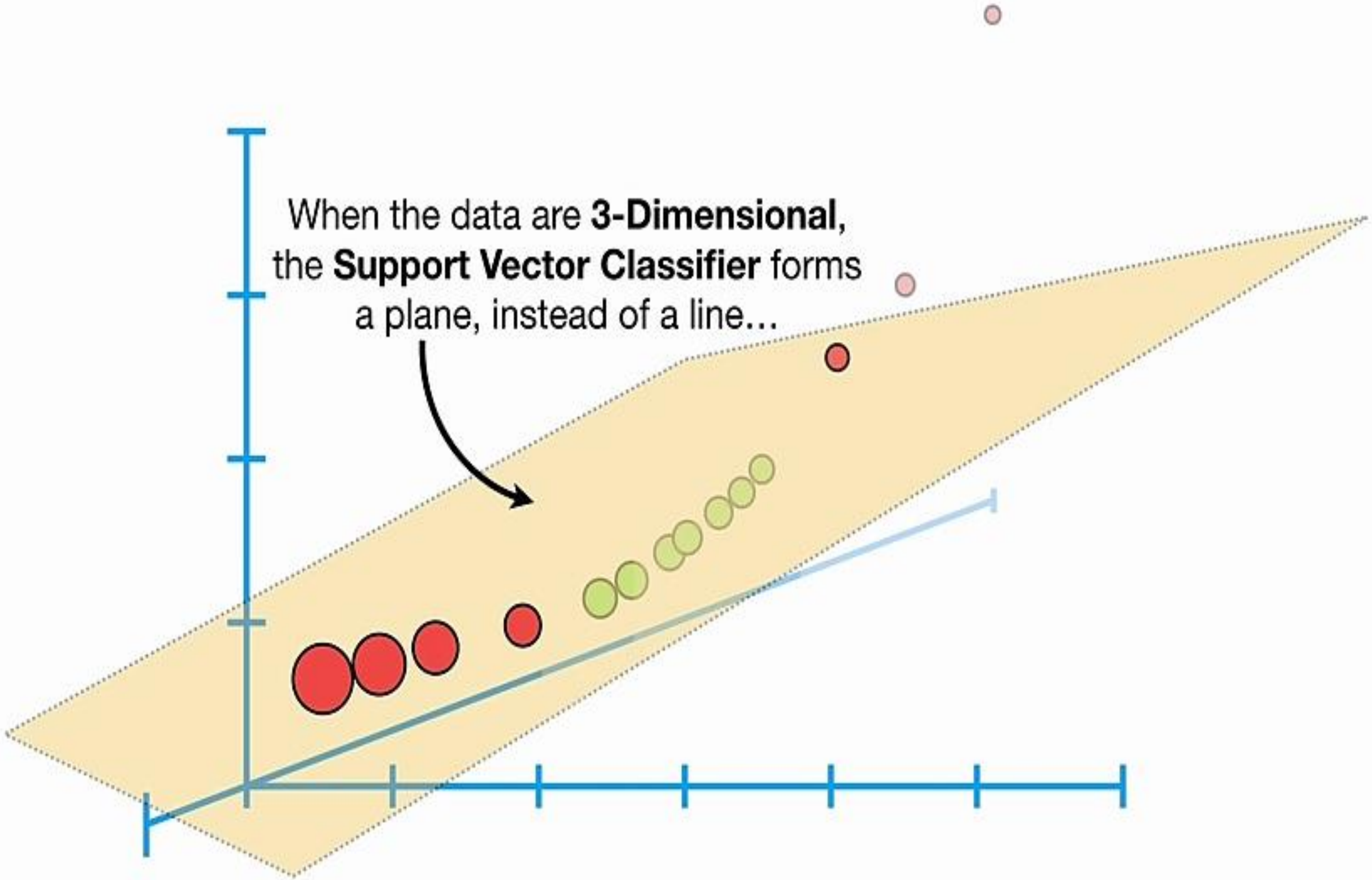


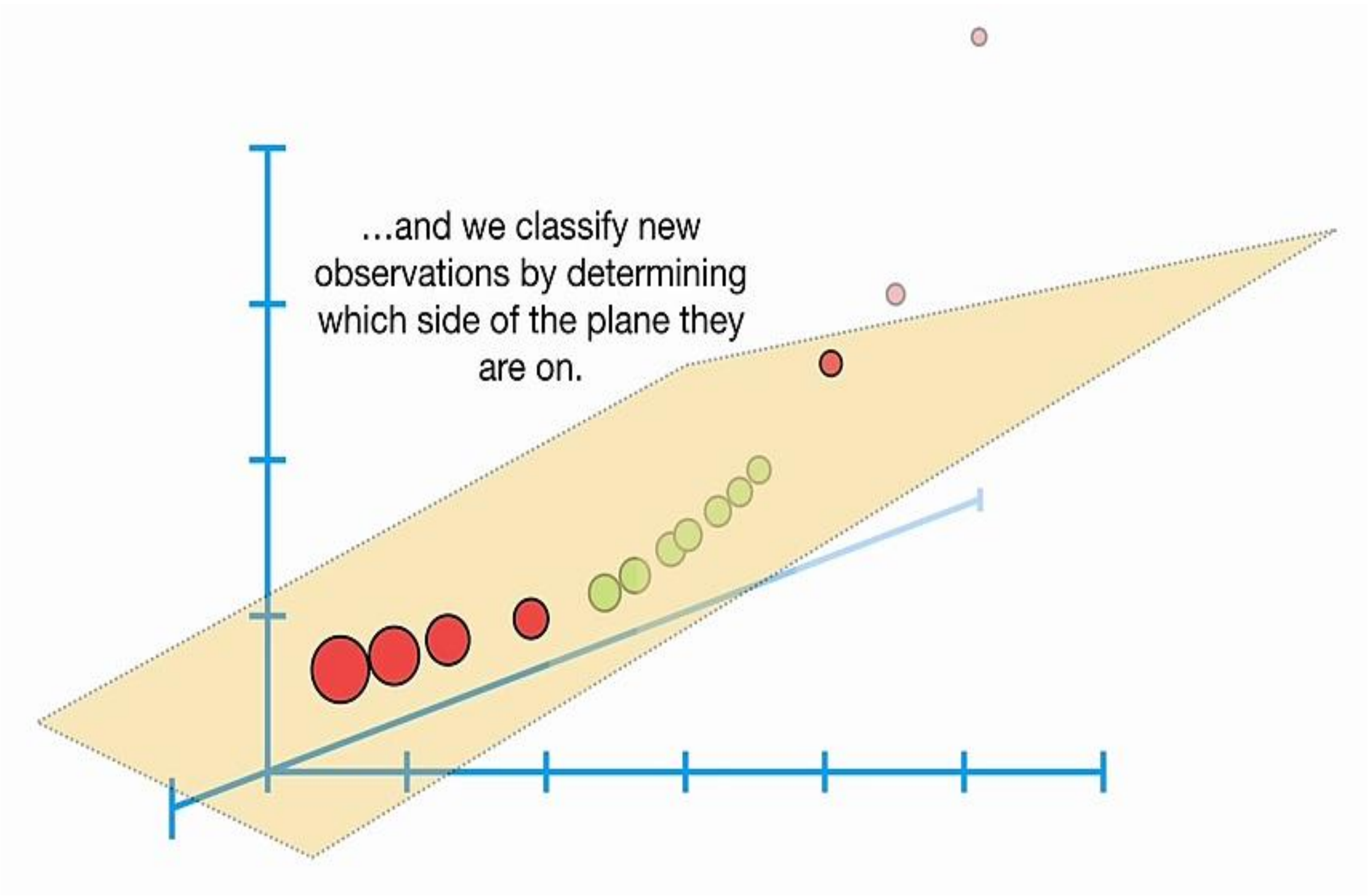
...then the data would
be **3-Dimensional**.
Height (cm):
Age
Mass (g):

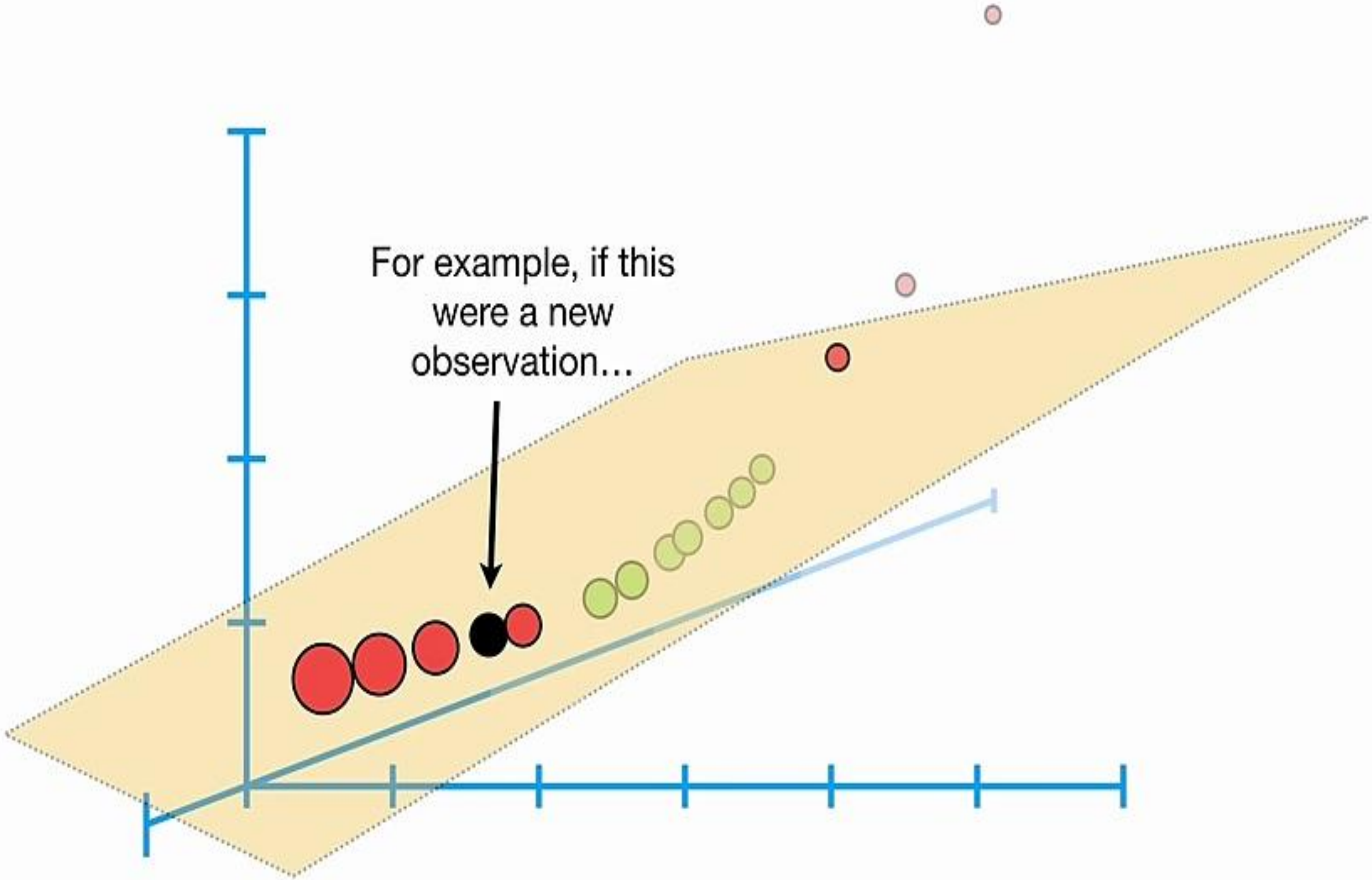

NOTE: The axis that ***age*** is on is supposed to represent ***depth***…
Height (cm):
Age
Mass (g):

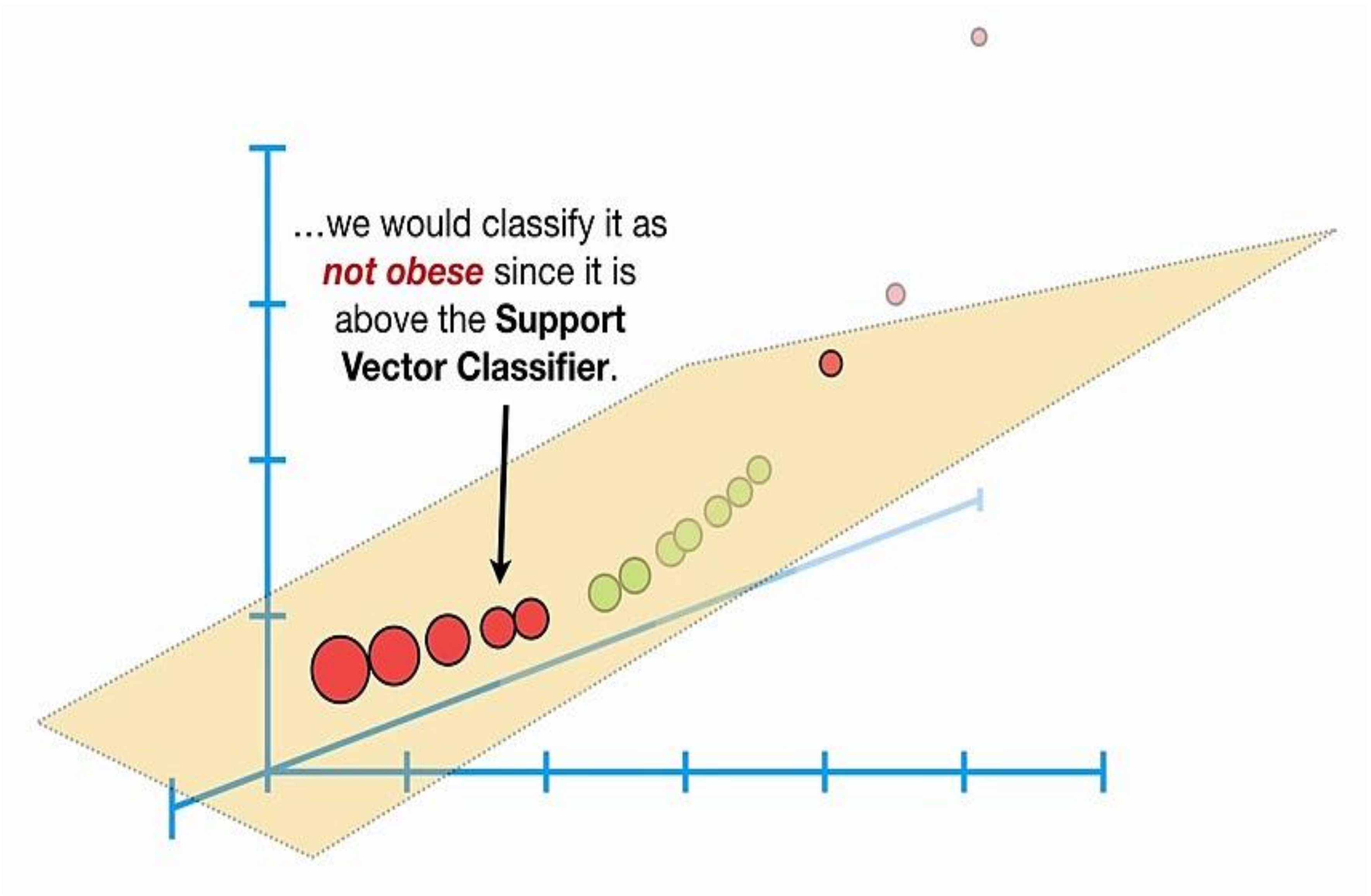

...and these circles are larger
in order to to appear closer,
and thus younger...
Height (cm):
Age
Mass (g):


...and these circles are
smaller in order to look
further away, and thus, older.
Height (cm):
Age
Mass (g):

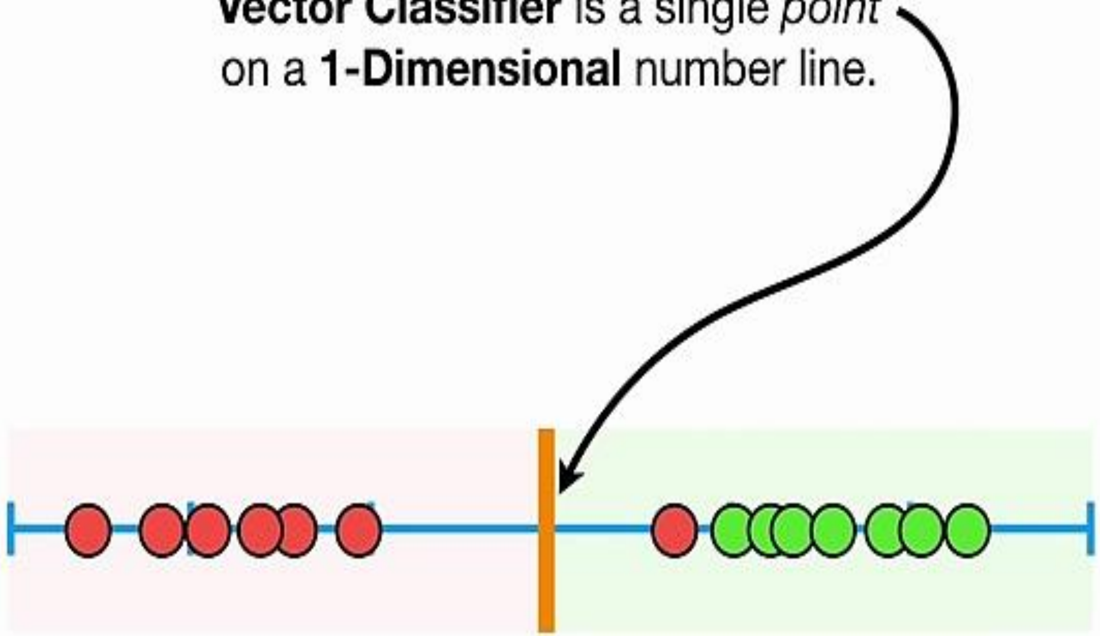

When the data are **3-Dimensional**,
the **Support Vector Classifier** forms
a plane, instead of a line…

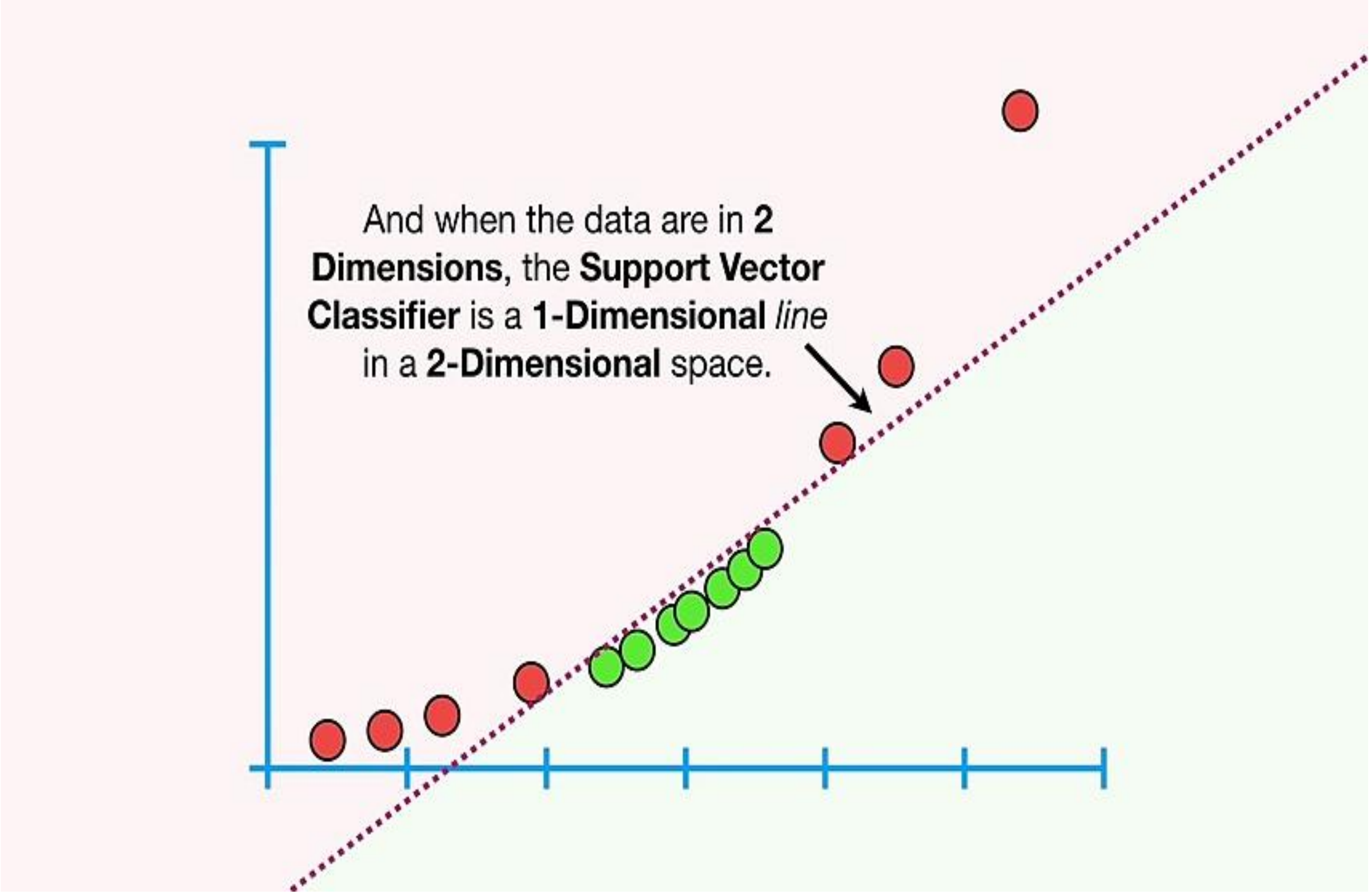

...and we classify new observations by determining which side of the plane they are on.

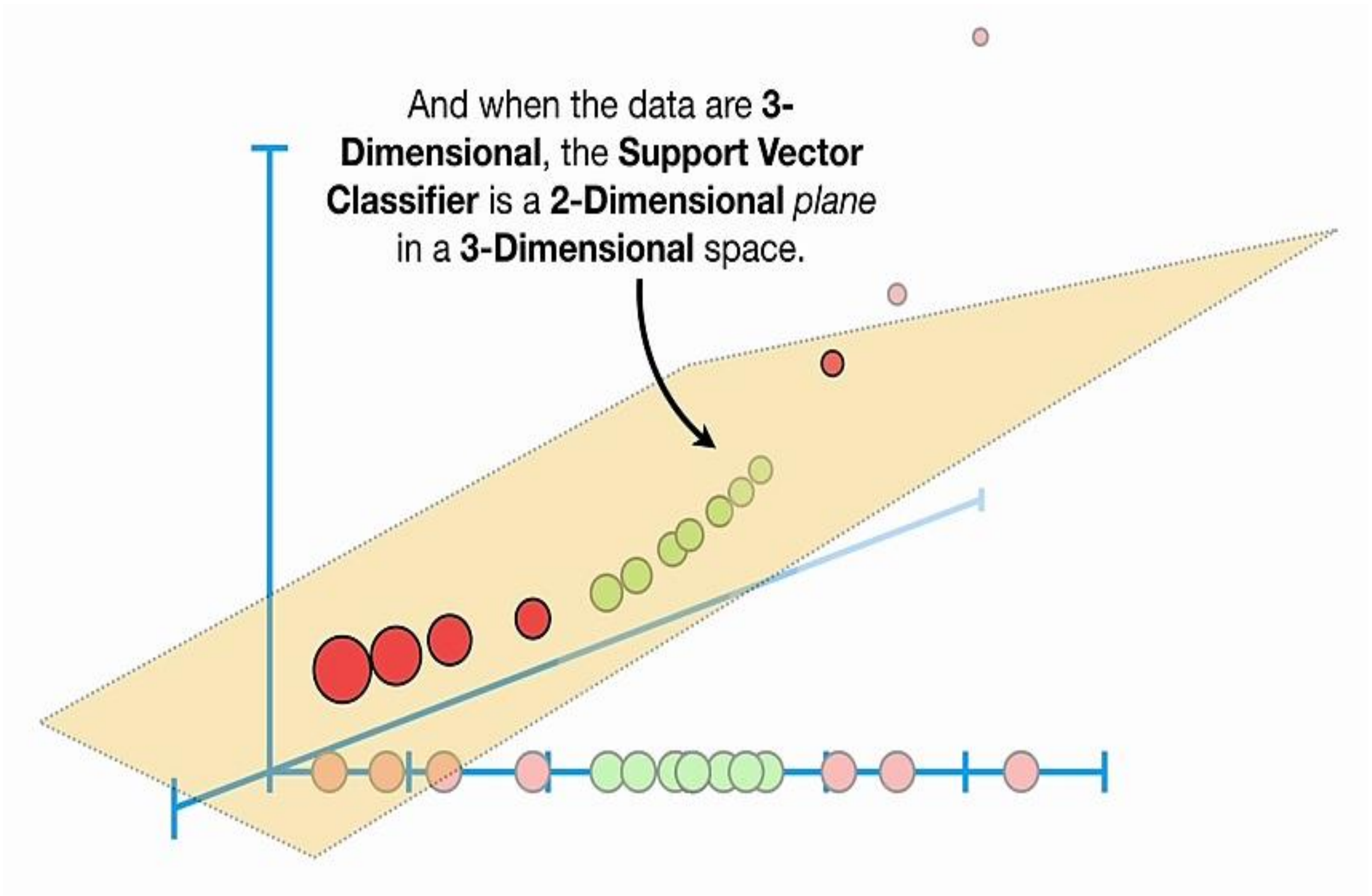

For example, if this
were a new
observation...


...we would classify it as
not obese since it is
above the **Support
Vector Classifier**.

But we know that when the data are **1-Dimensional**, the **Support Vector Classifier** is a single *point* on a **1-Dimensional** number line.



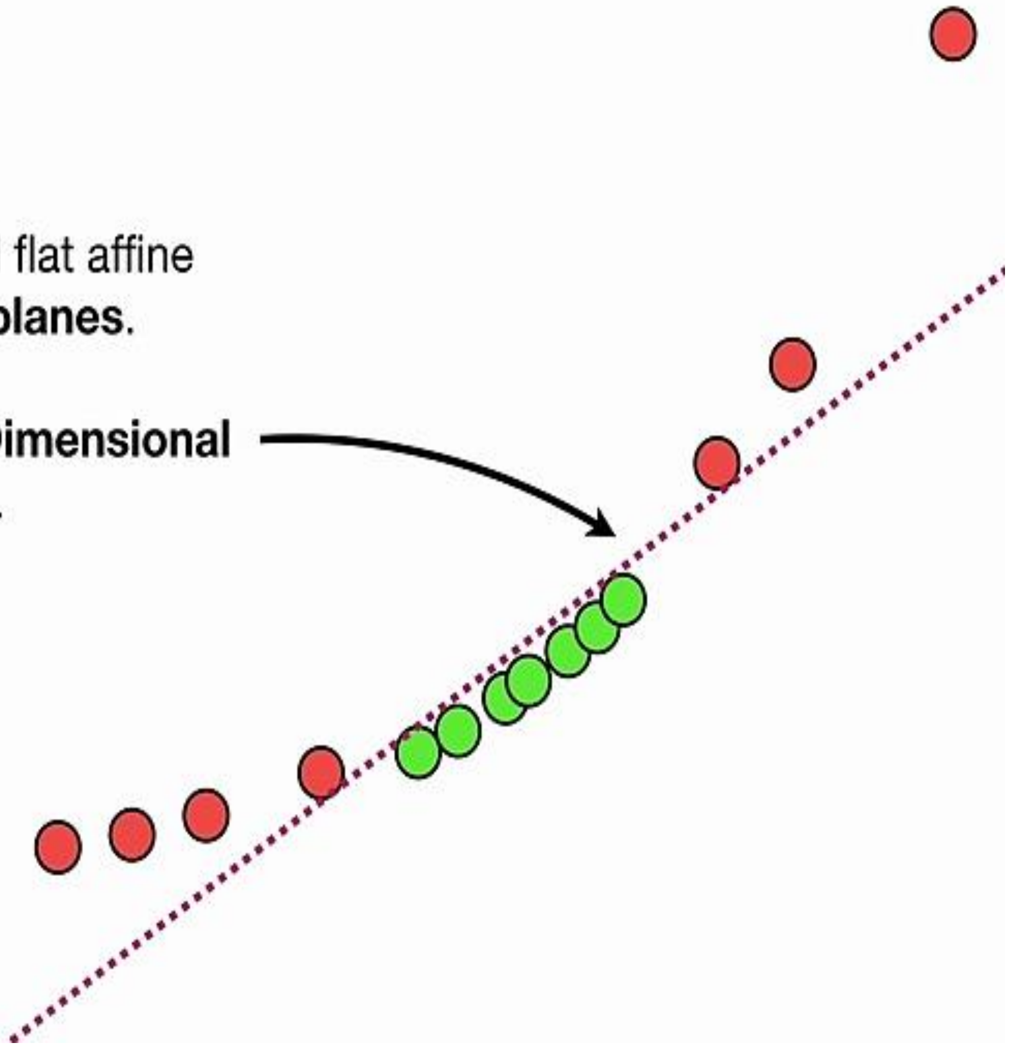

And when the data are in **2 Dimensions**, the **Support Vector Classifier** is a **1-Dimensional** *line* in a **2-Dimensional** space.


And when the data are **3-Dimensional**, the **Support Vector Classifier** is a **2-Dimensional** *plane* in a **3-Dimensional** space.

And when the data are in **4 or more Dimensions**, the **Support Vector Classifier** is a *hyperplane*.

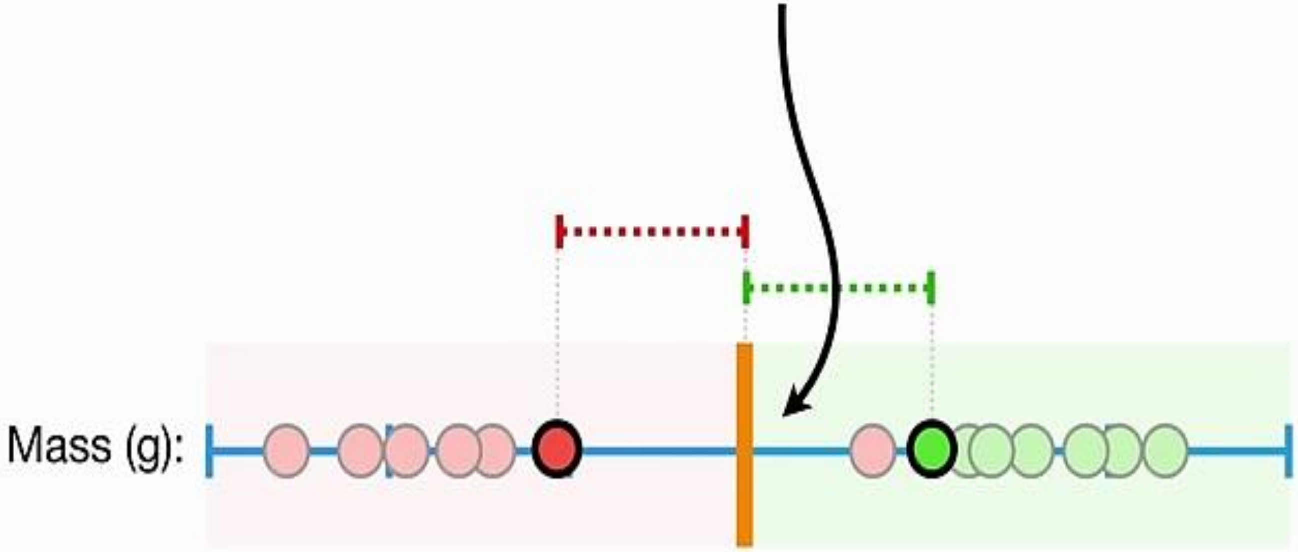
NOTE: Technically speaking, all flat affine subspaces are called **hyperplanes**.

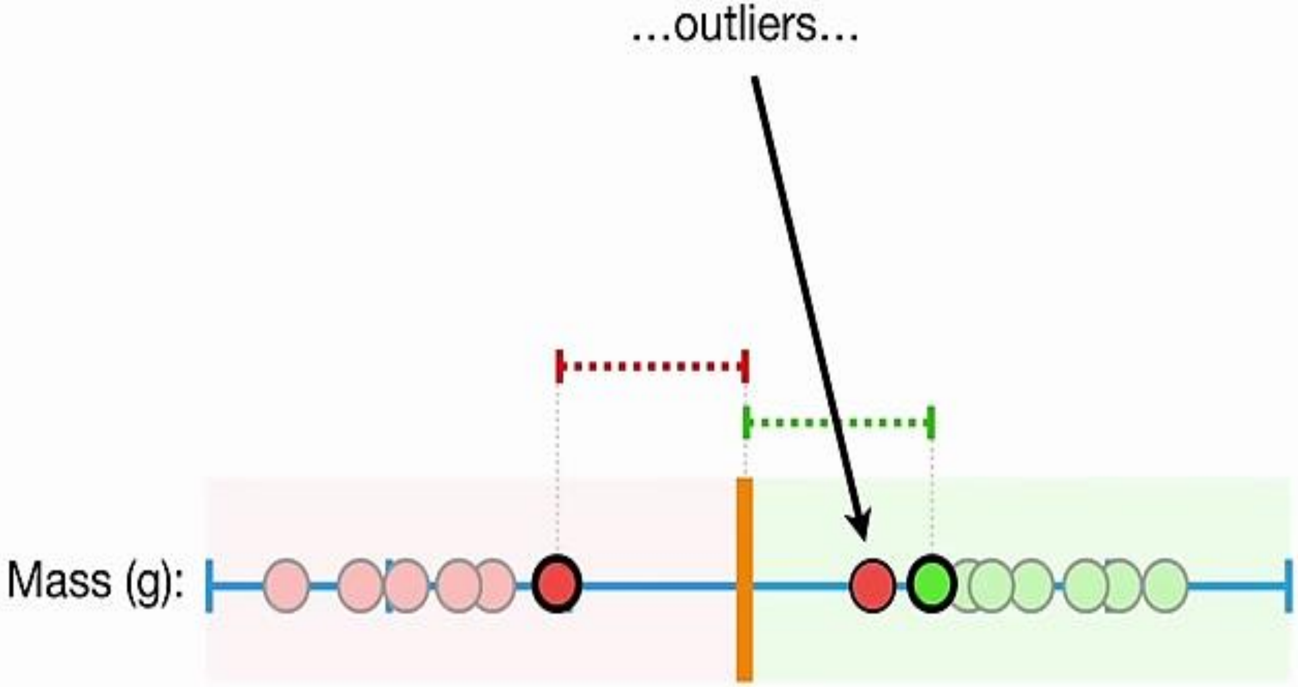
So, technically speaking, this **1-Dimensional** *line* is a **hyperplane**…



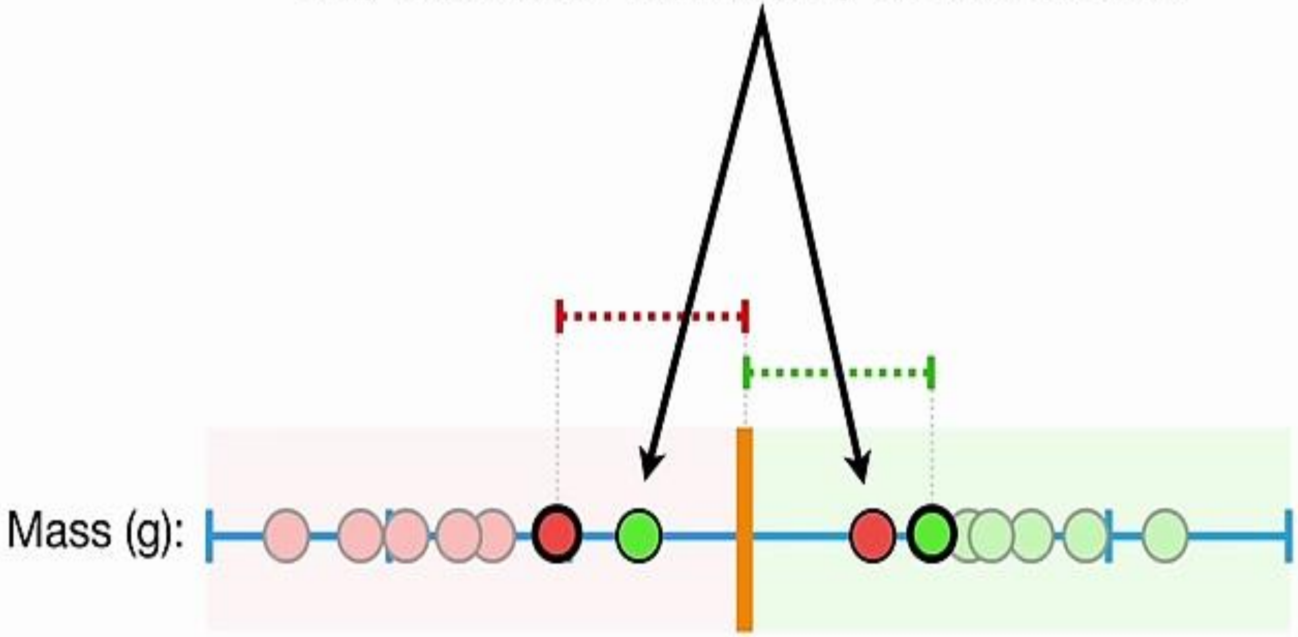
Non-linear SVM

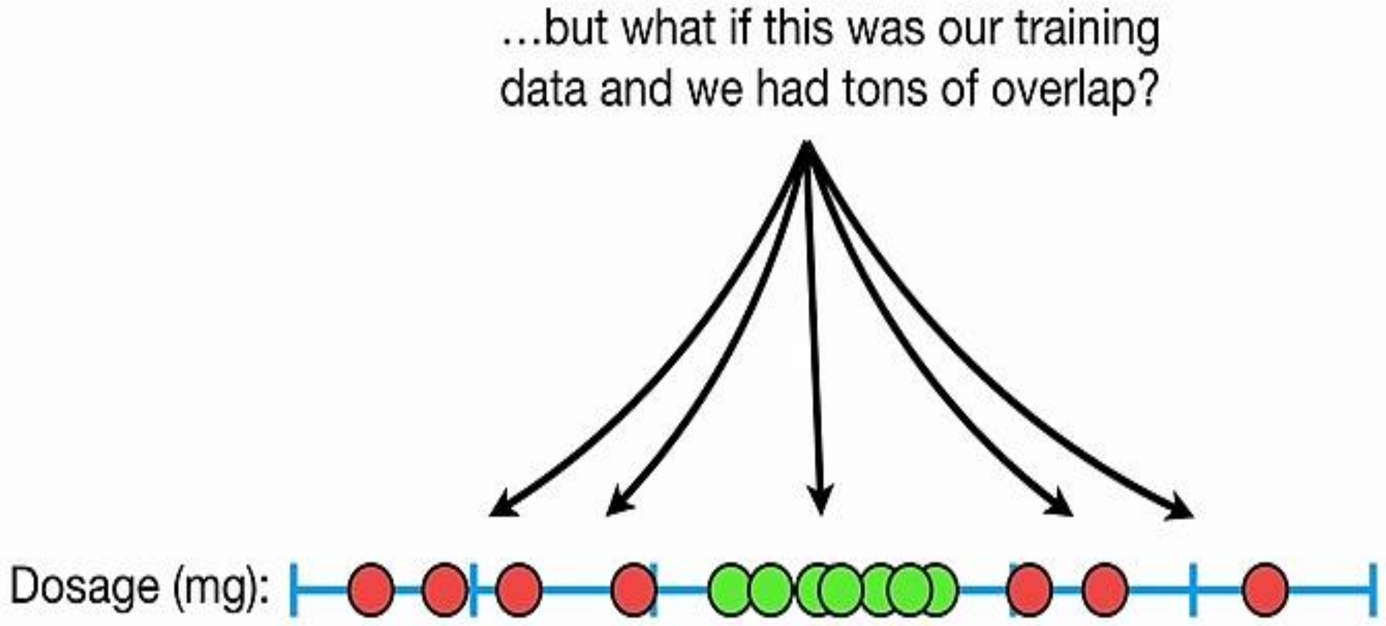
Support Vector Classifiers seem pretty cool because they can handle...



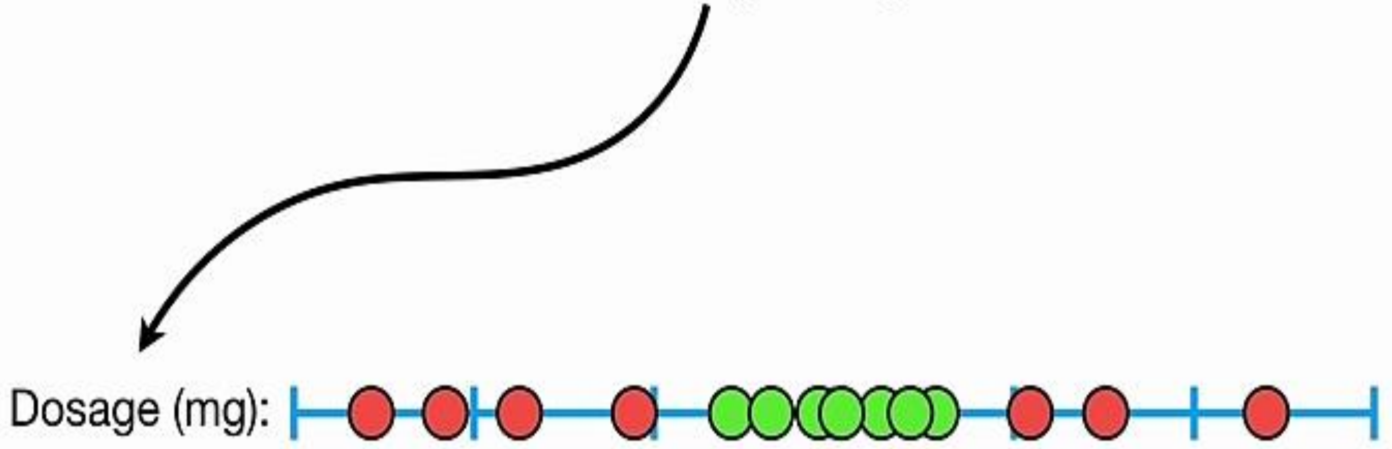

...outliers...
Mass (g):

…and, because they allow misclassifications, they can handle overlapping classifications…

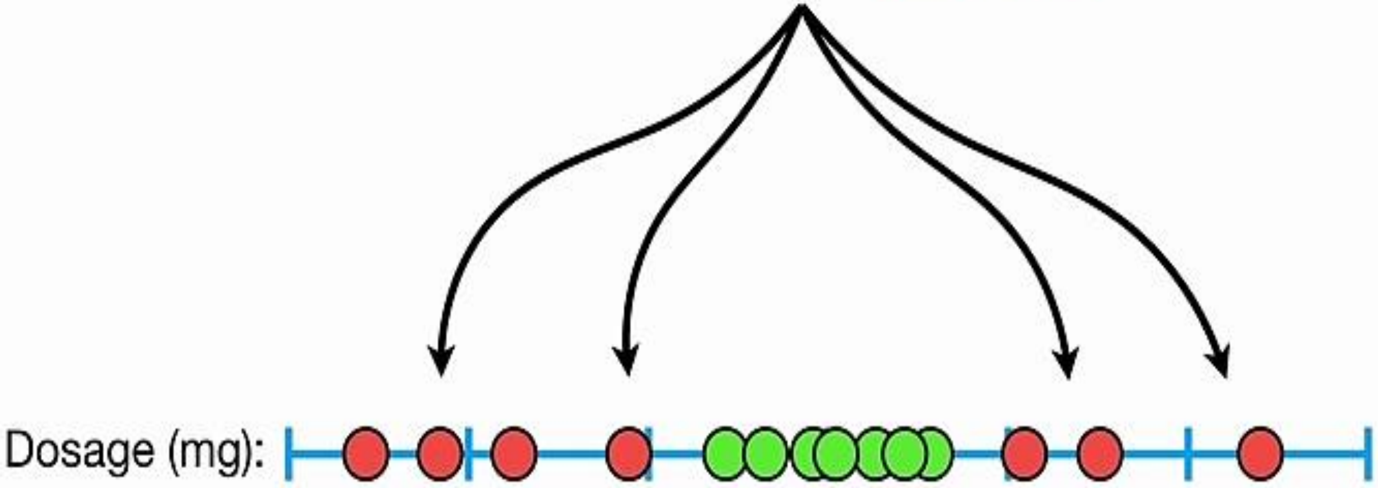



...but what if this was our training
data and we had tons of overlap?
Dosage (mg):

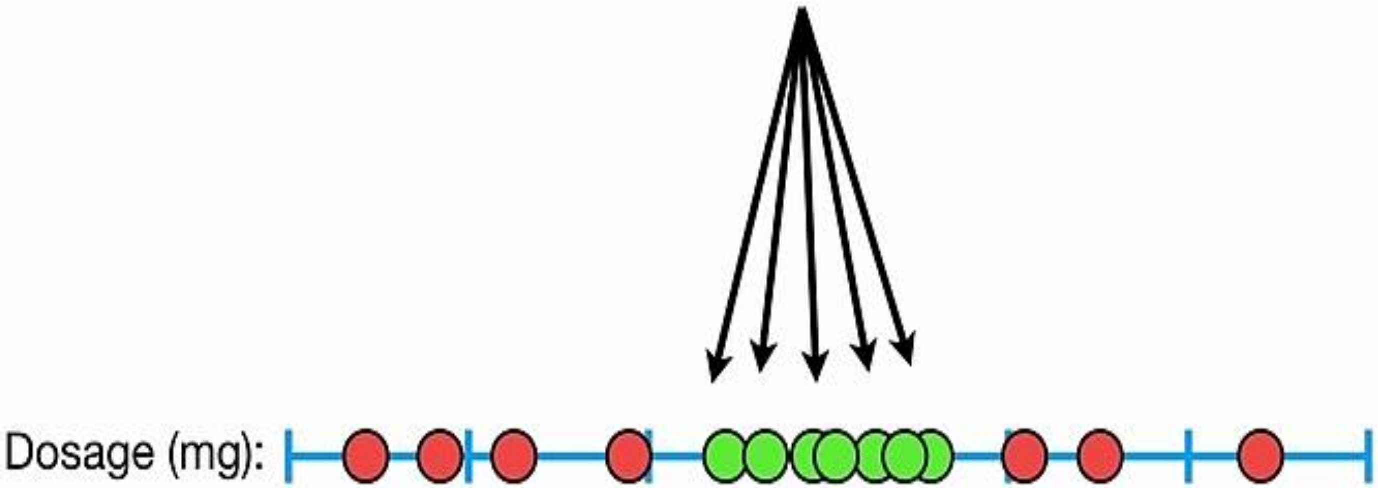
In this new example, with tons of overlap, we are now looking at **Drug Dosages**...



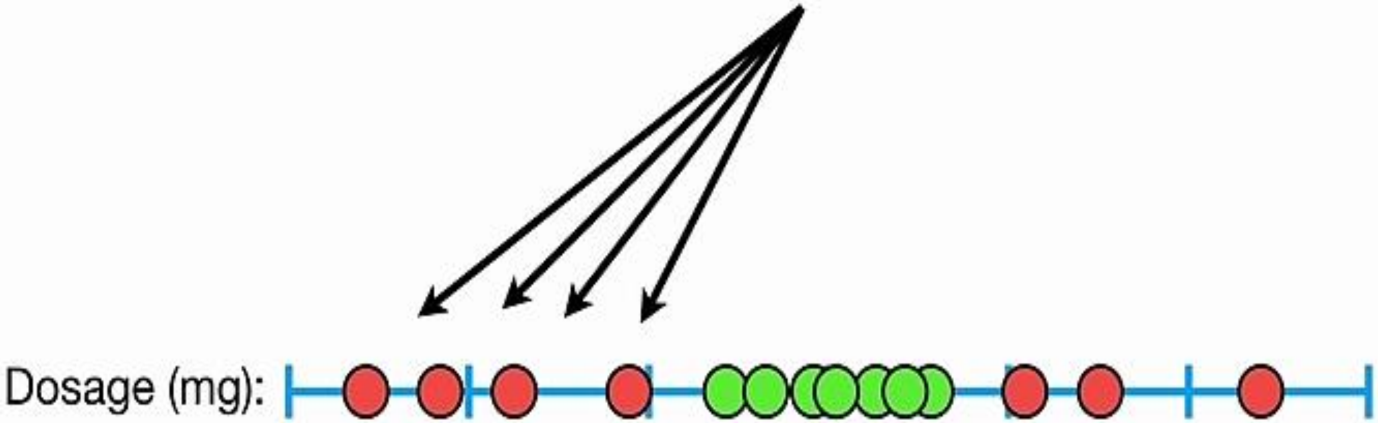
…and the **red dots** represent
patients that were ***not cured***…

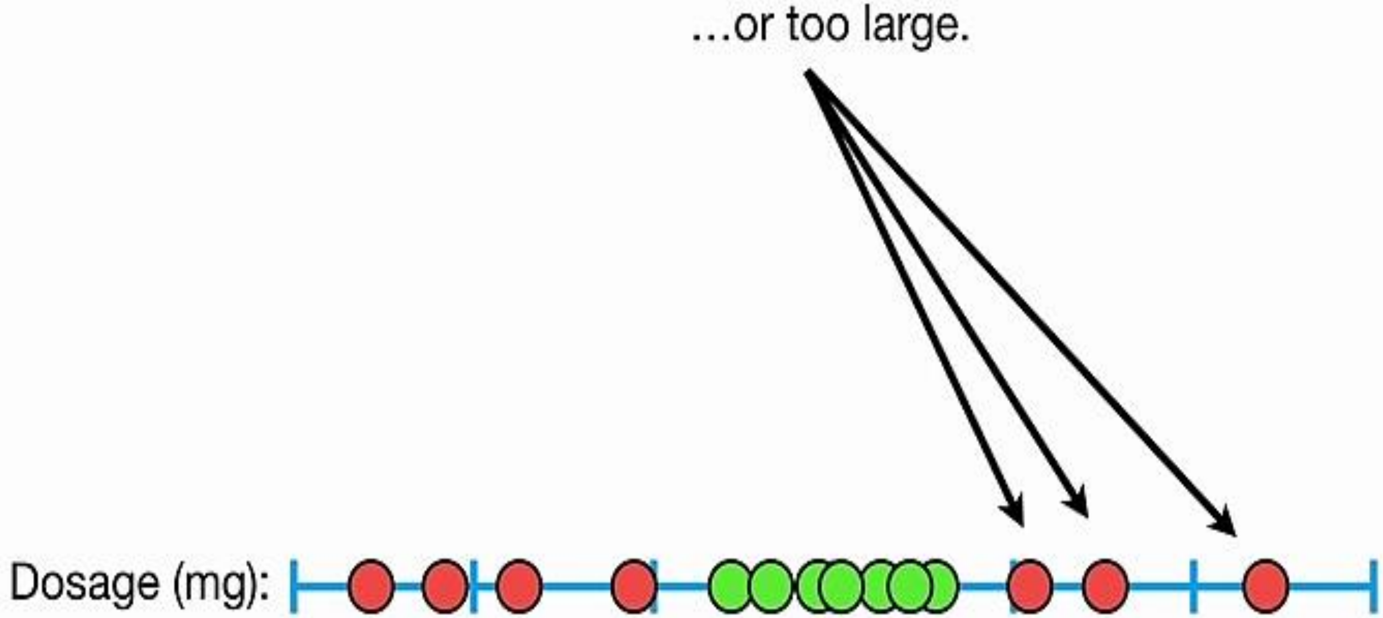


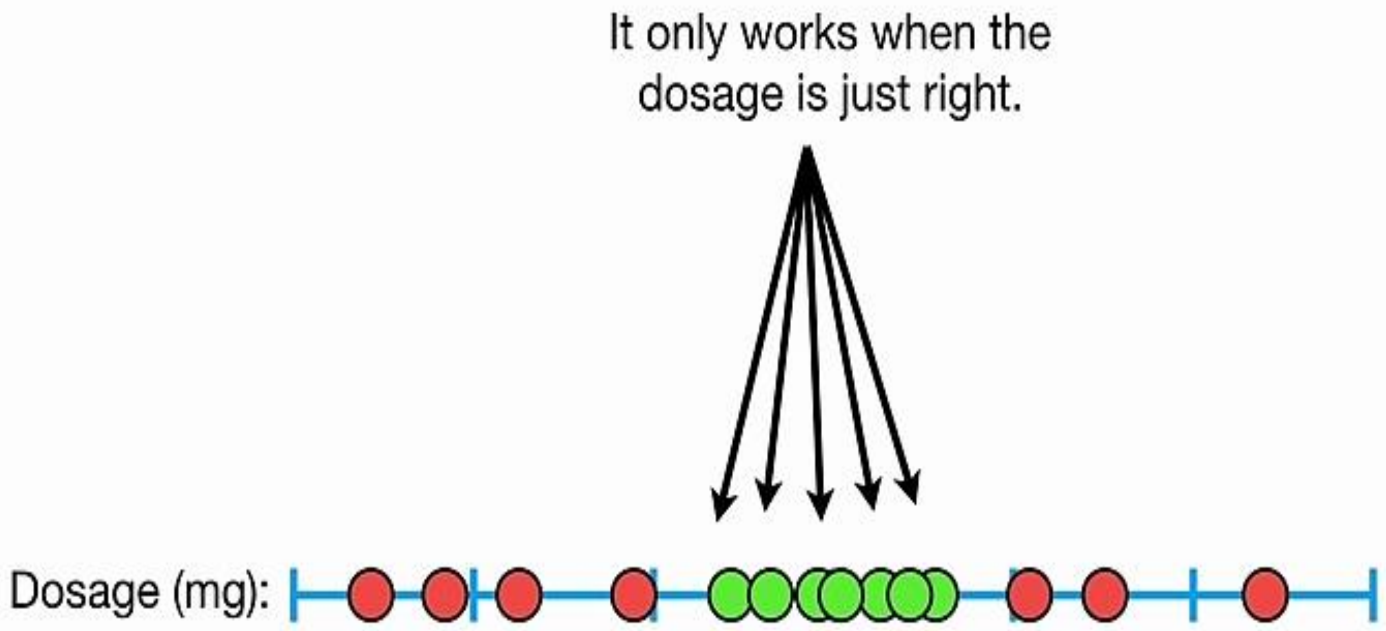
...and the **green dots** represent patients that were ***cured***.



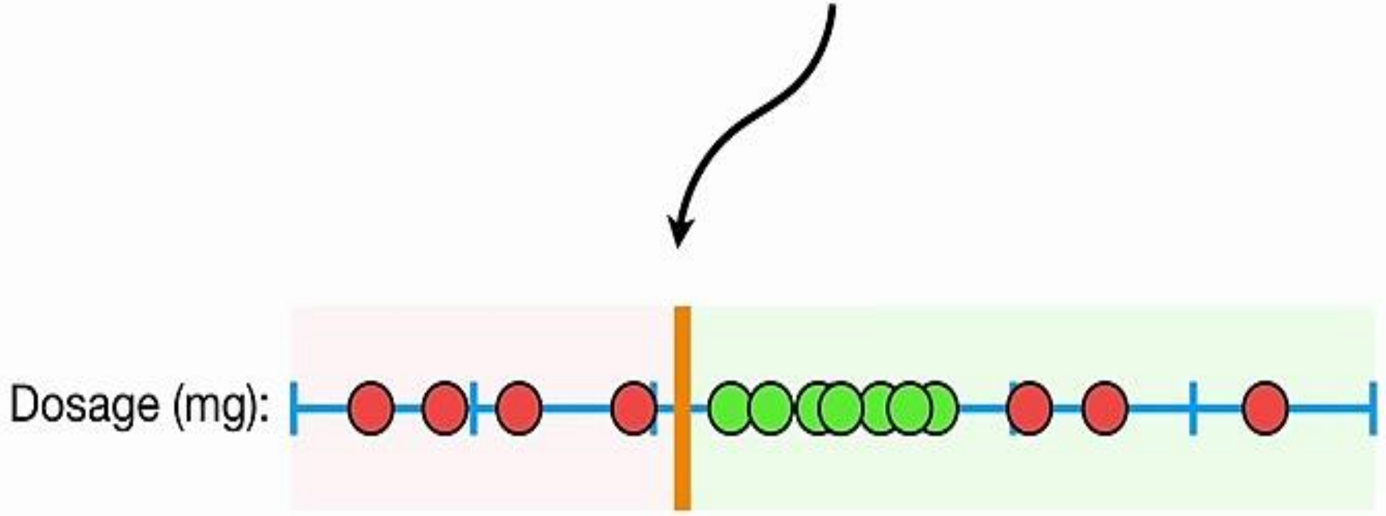
In other words, the drug doesn't work if the dosage is too small…



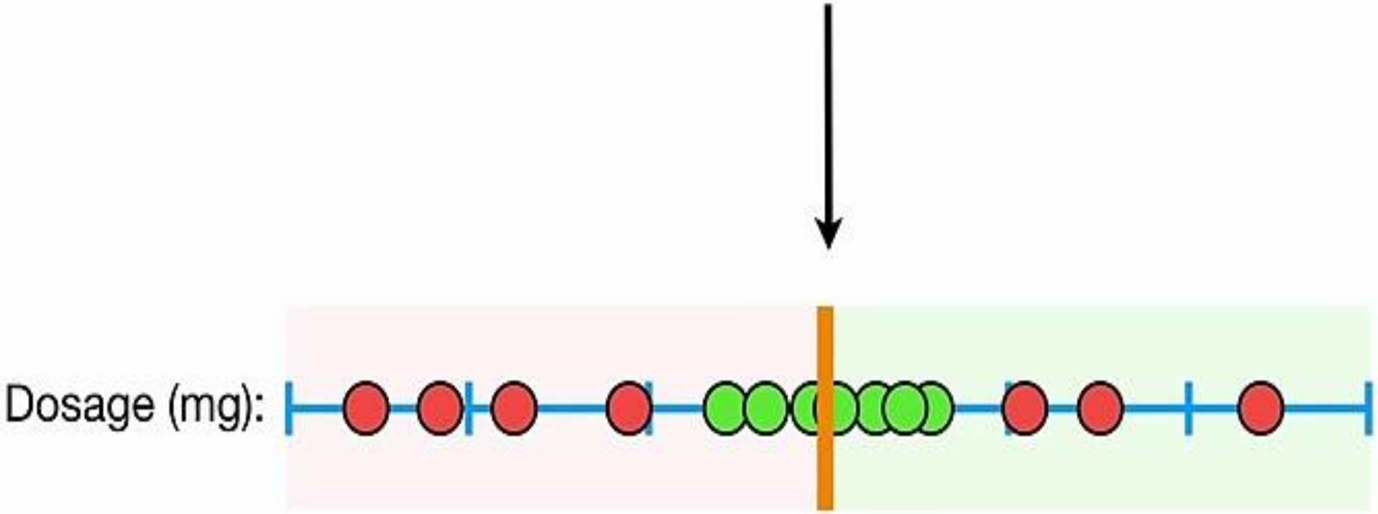

…or too large.
Dosage (mg):

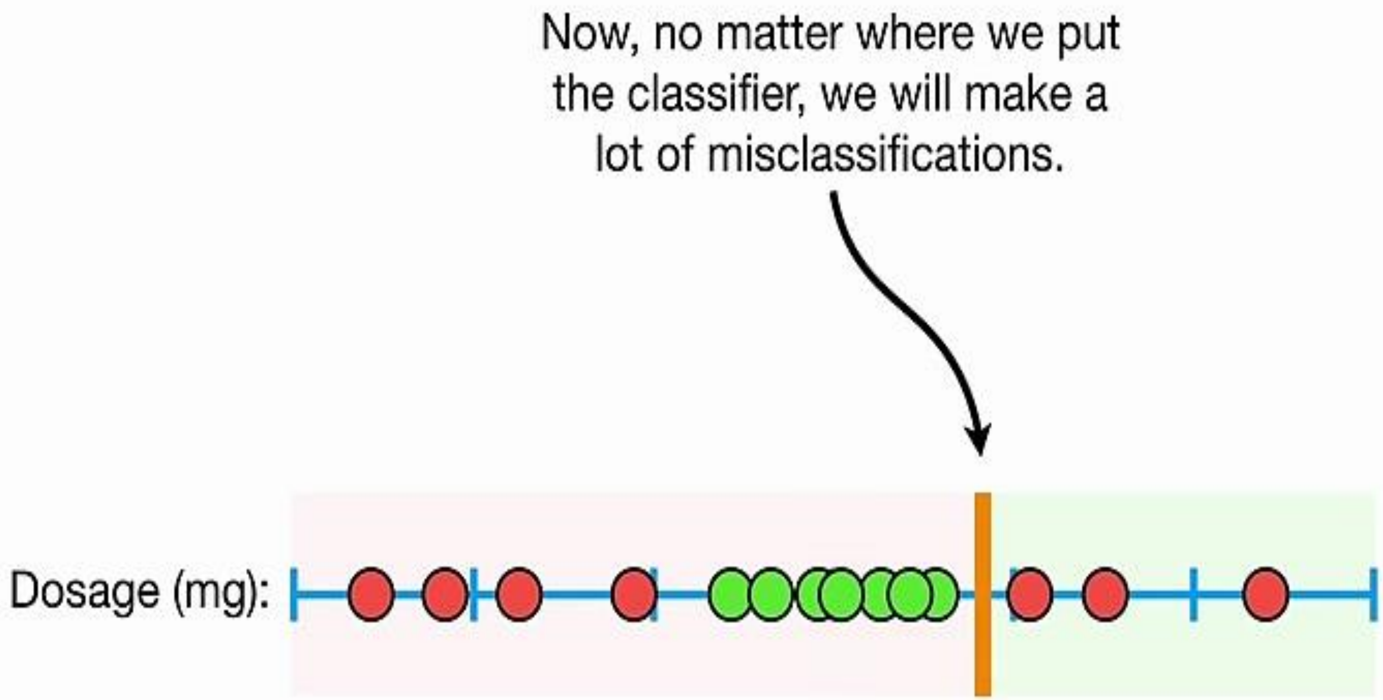

It only works when the dosage is just right.
Dosage (mg):

Now, no matter where we put the classifier, we will make a lot of misclassifications.

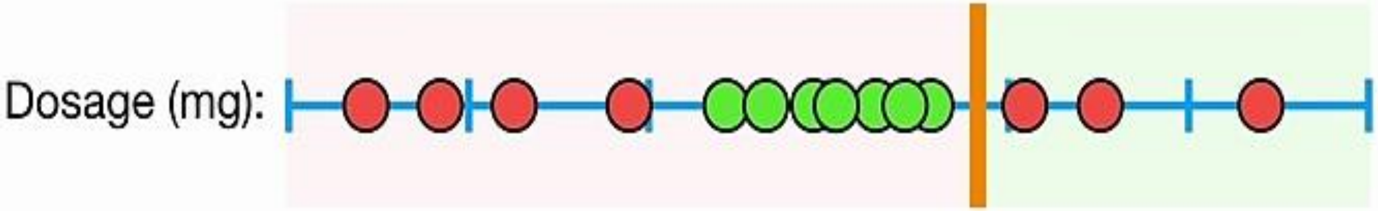


Now, no matter where we put the classifier, we will make a lot of misclassifications.

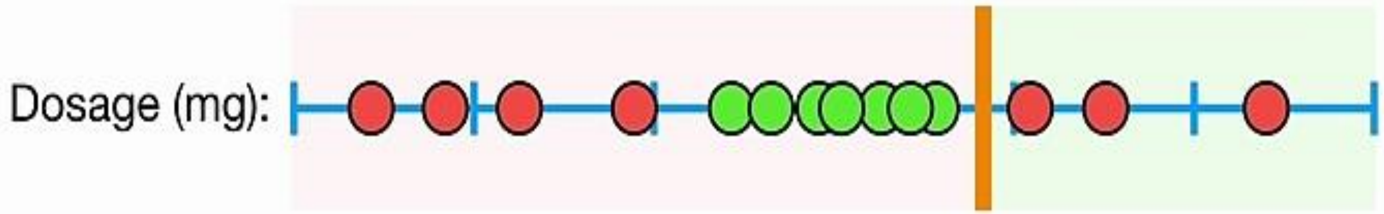


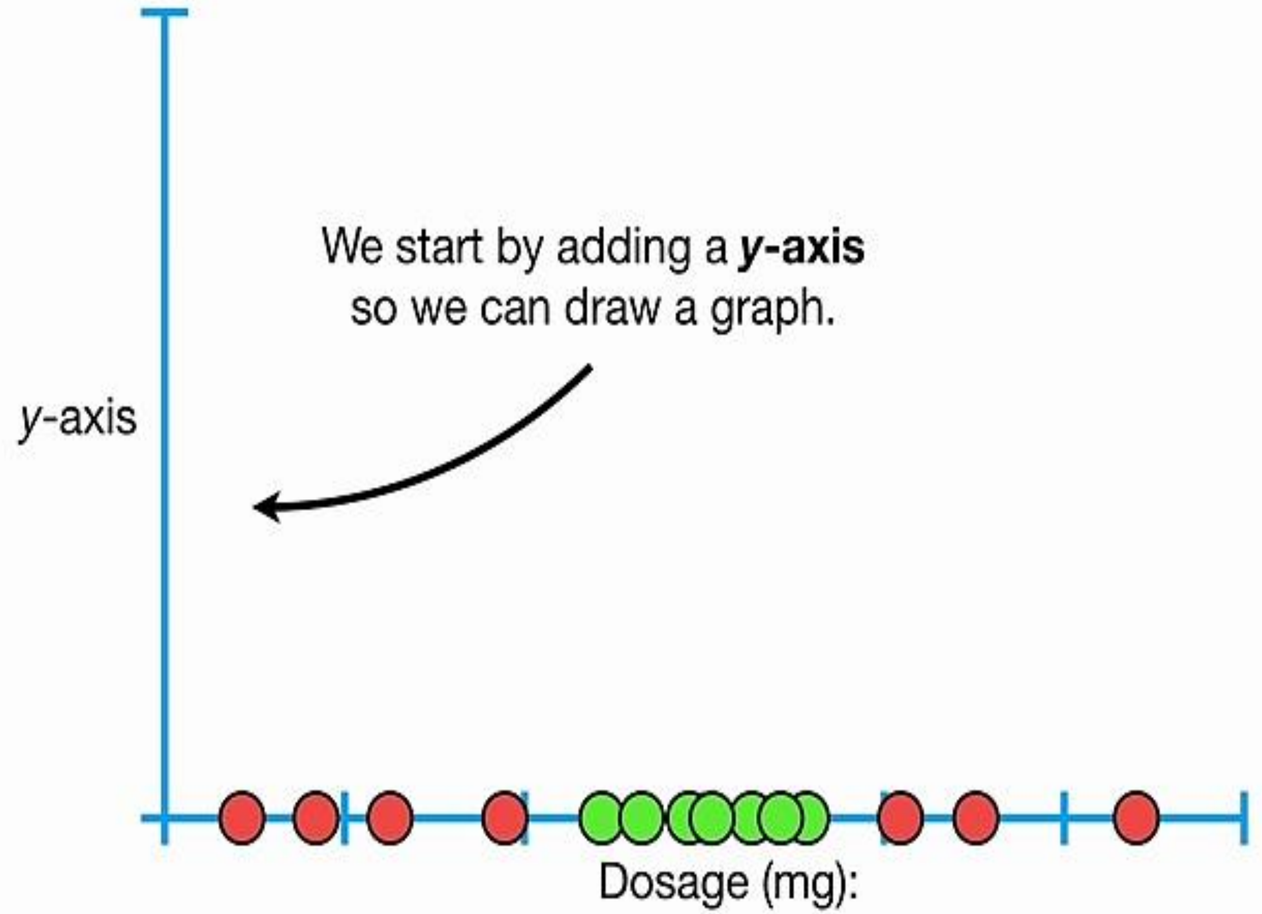

Now, no matter where we put the classifier, we will make a lot of misclassifications.
Dosage (mg):

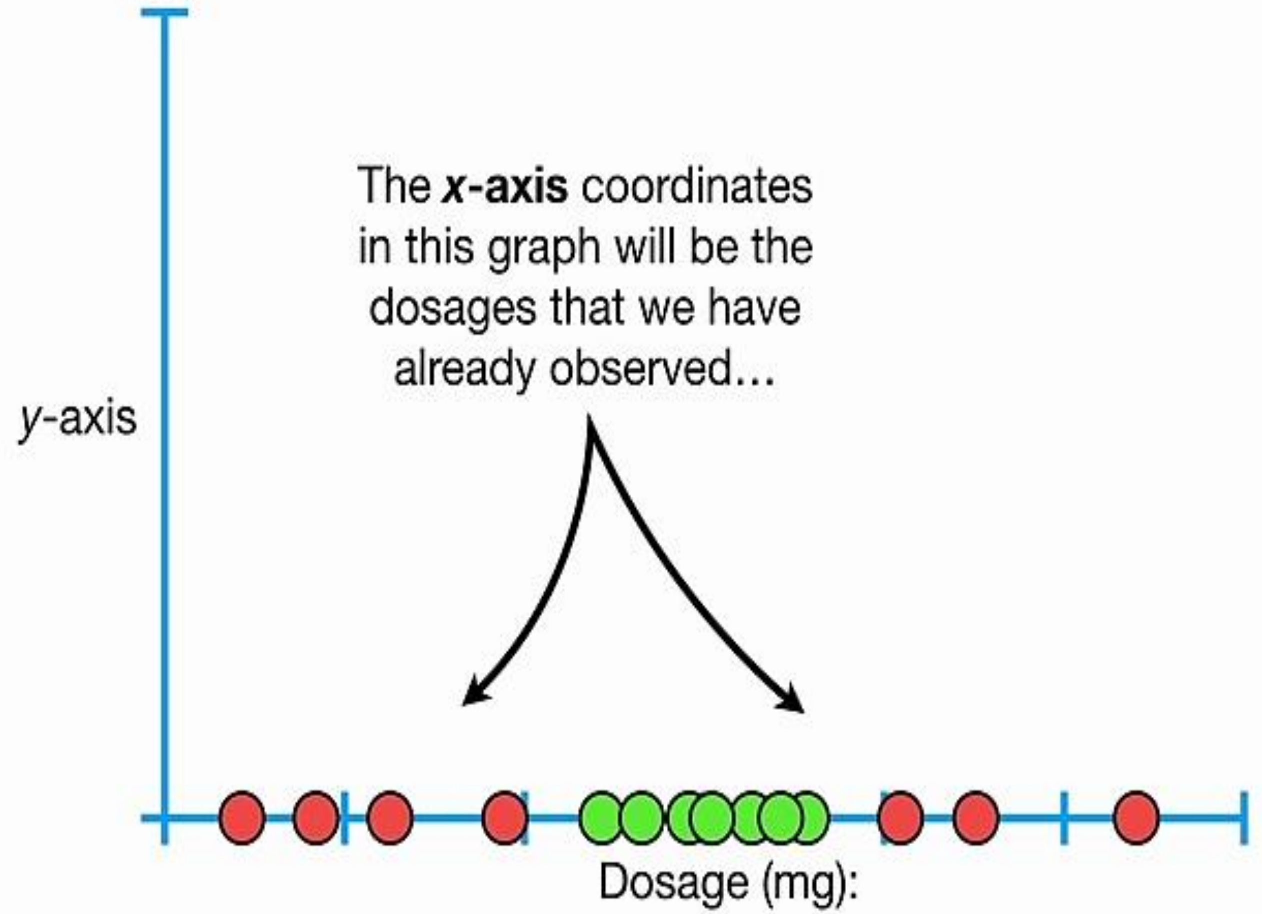
So **Support Vector Classifiers** are are only semi-cool, since they don't perform well with this type of data.

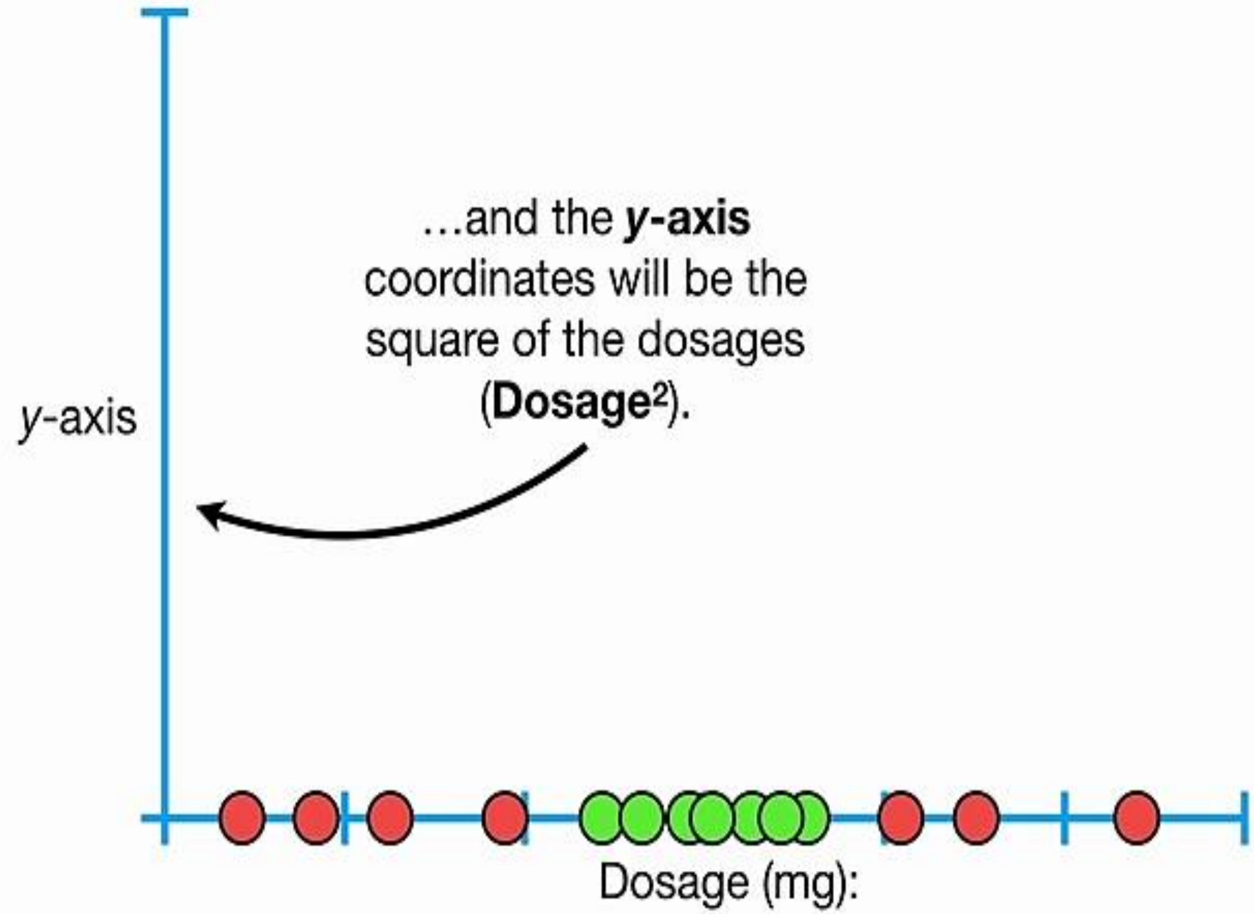


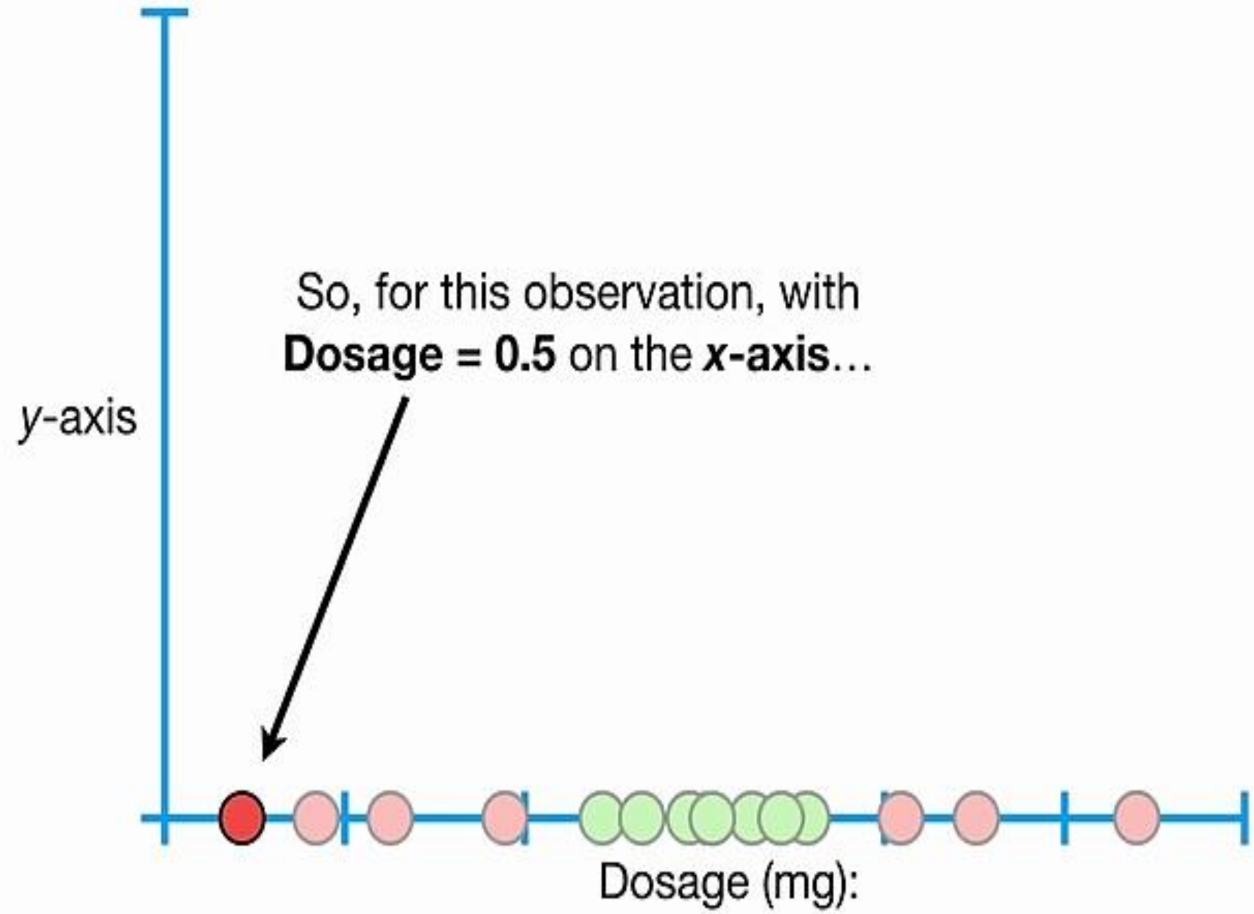
Can we do better than **Maximal Margin Classifiers** and **Support Vector Classifiers**?

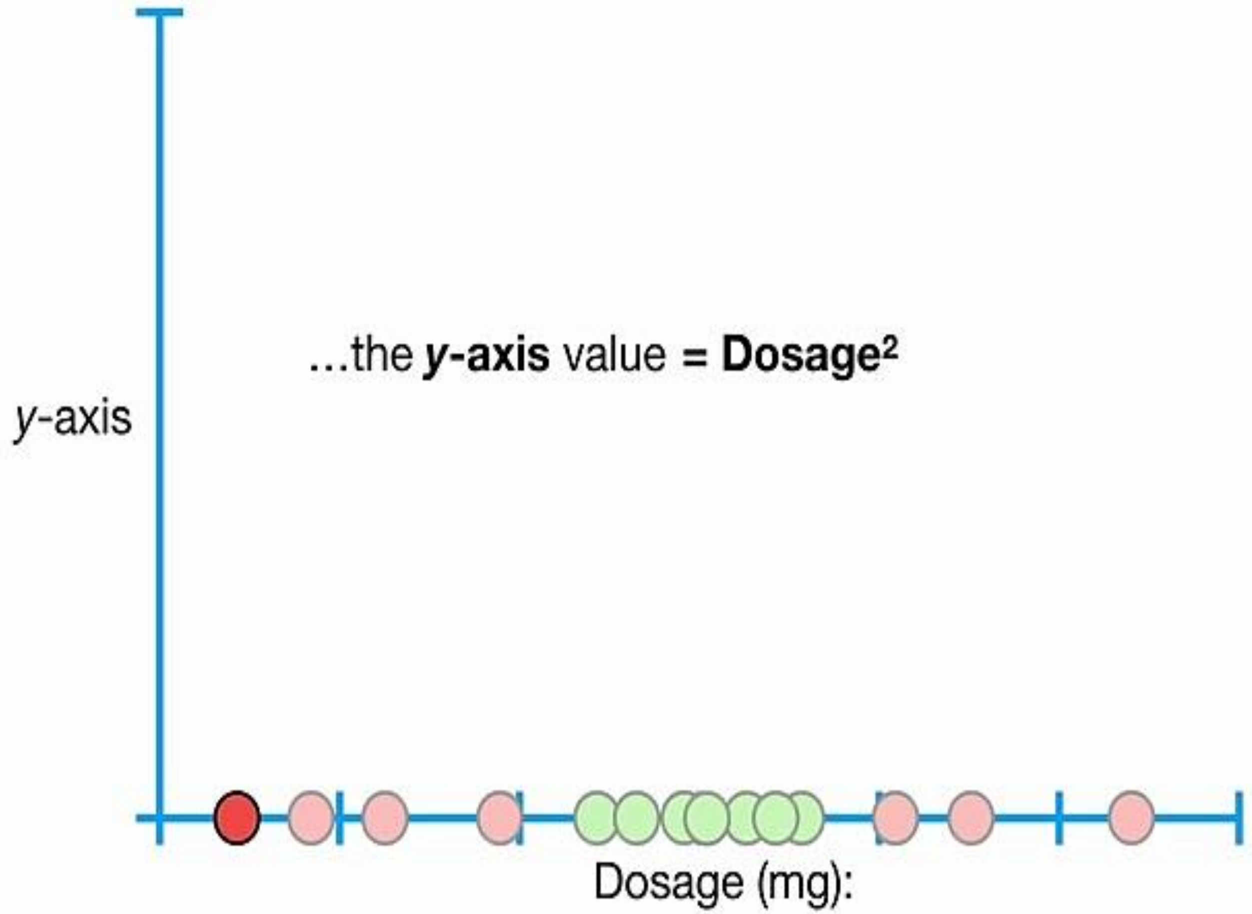


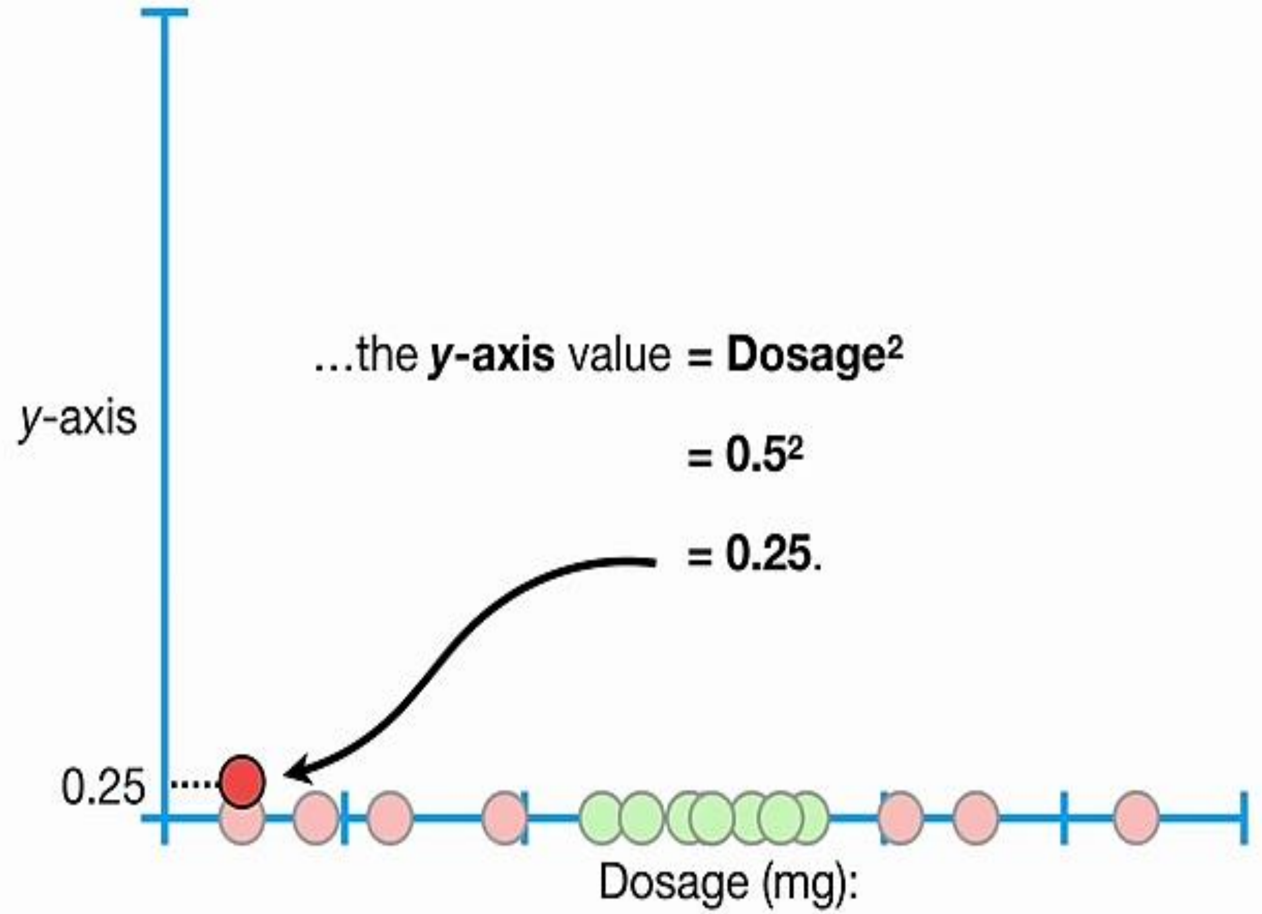

We start by adding a ***y*-axis**
so we can draw a graph.
y-axis
Dosage (mg):


The ***x*-axis** coordinates
in this graph will be the
dosages that we have
already observed...
y-axis
Dosage (mg):

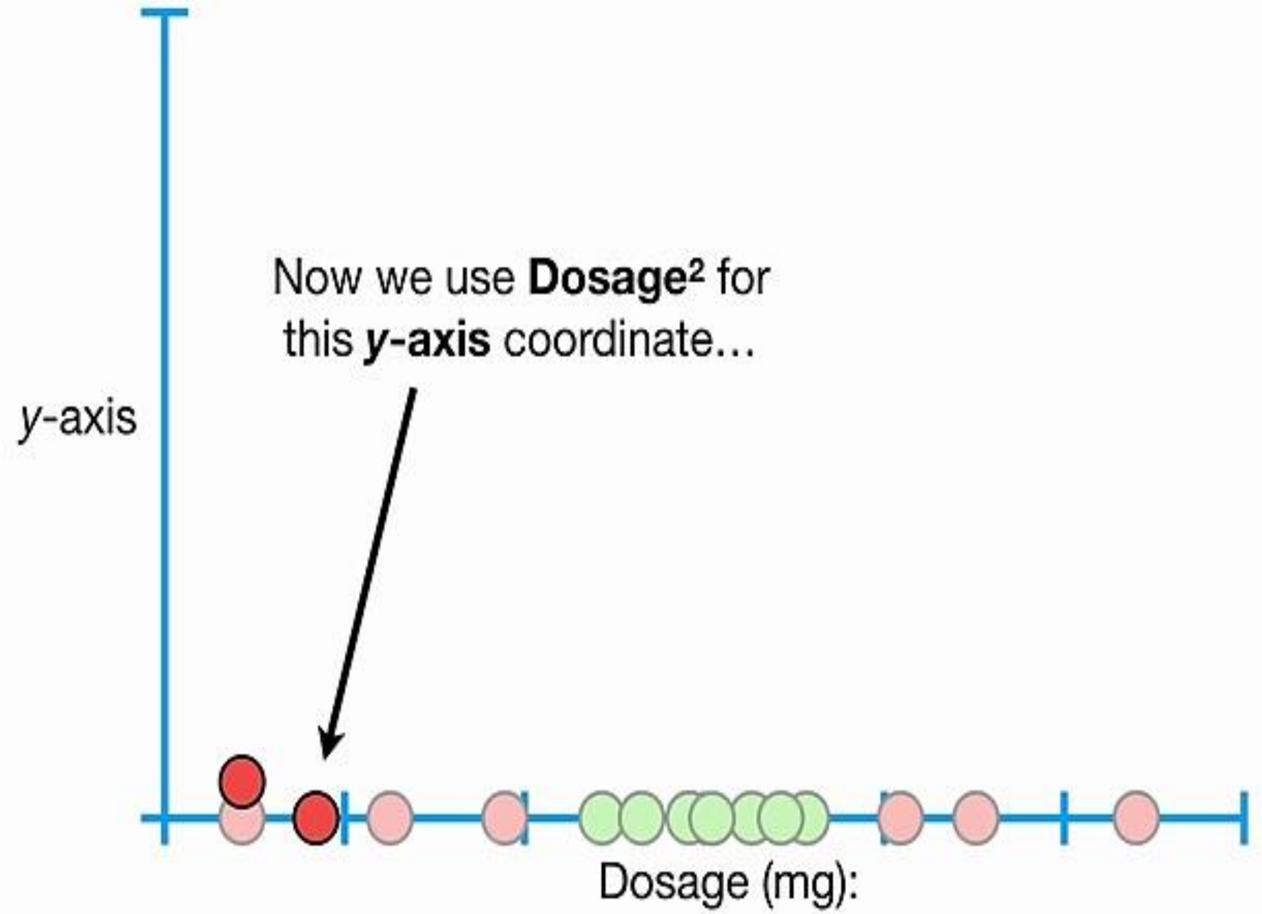

...and the ***y*-axis** coordinates will be the square of the dosages (**Dosage²**).
y-axis
Dosage (mg):

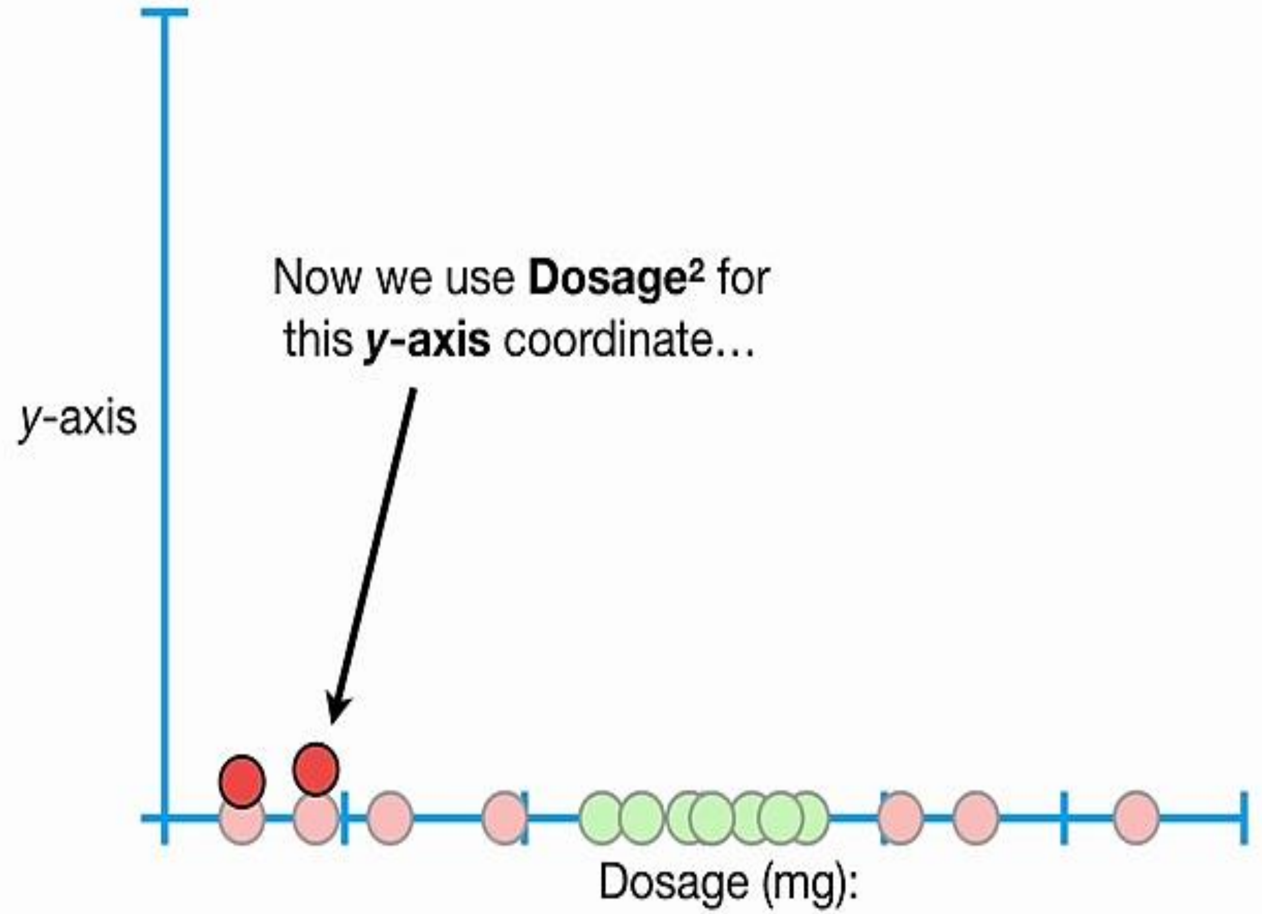

So, for this observation, with
Dosage = 0.5 on the ***x*-axis**…
y-axis
Dosage (mg):

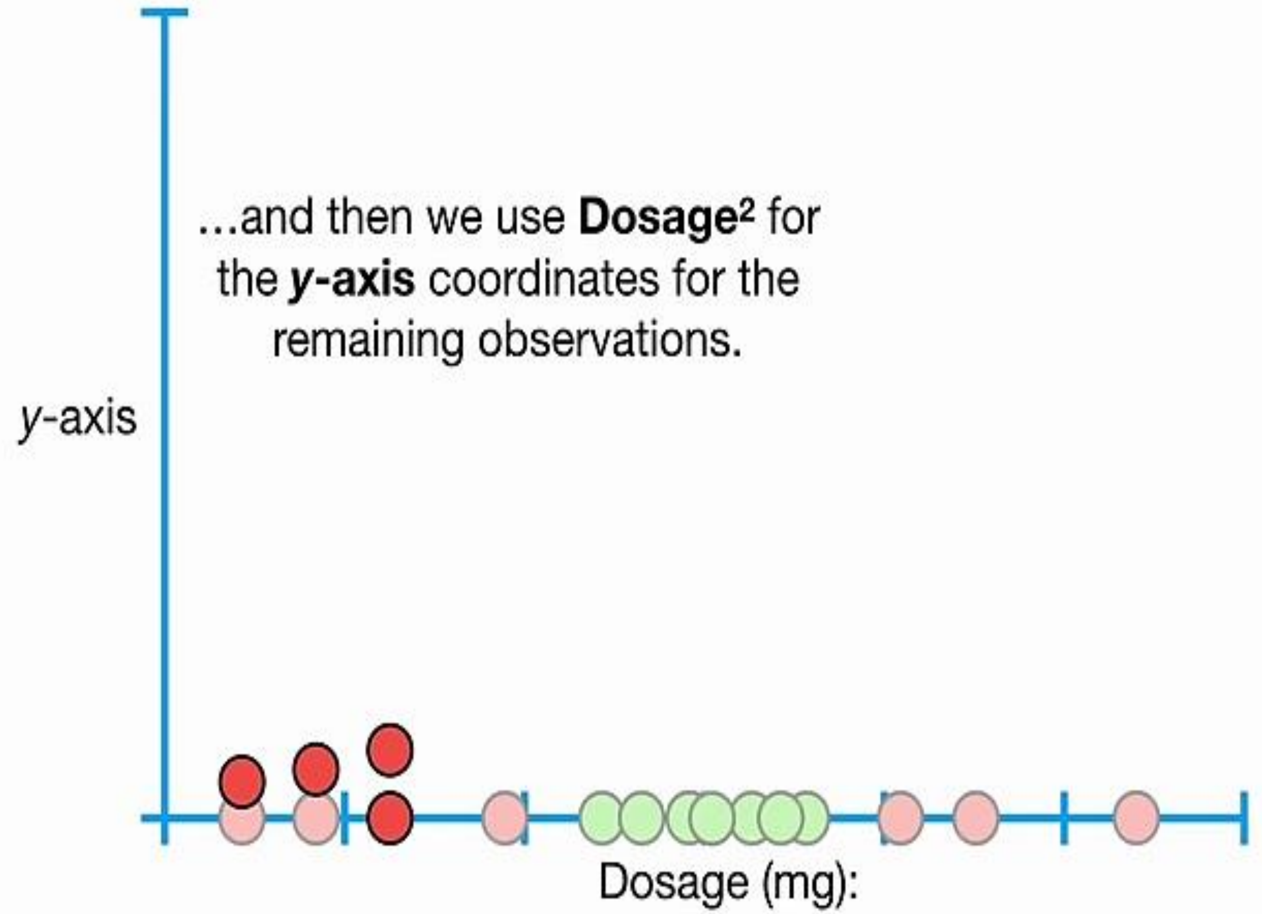

...the **y-axis** value = **Dosage²**
y-axis
Dosage (mg):

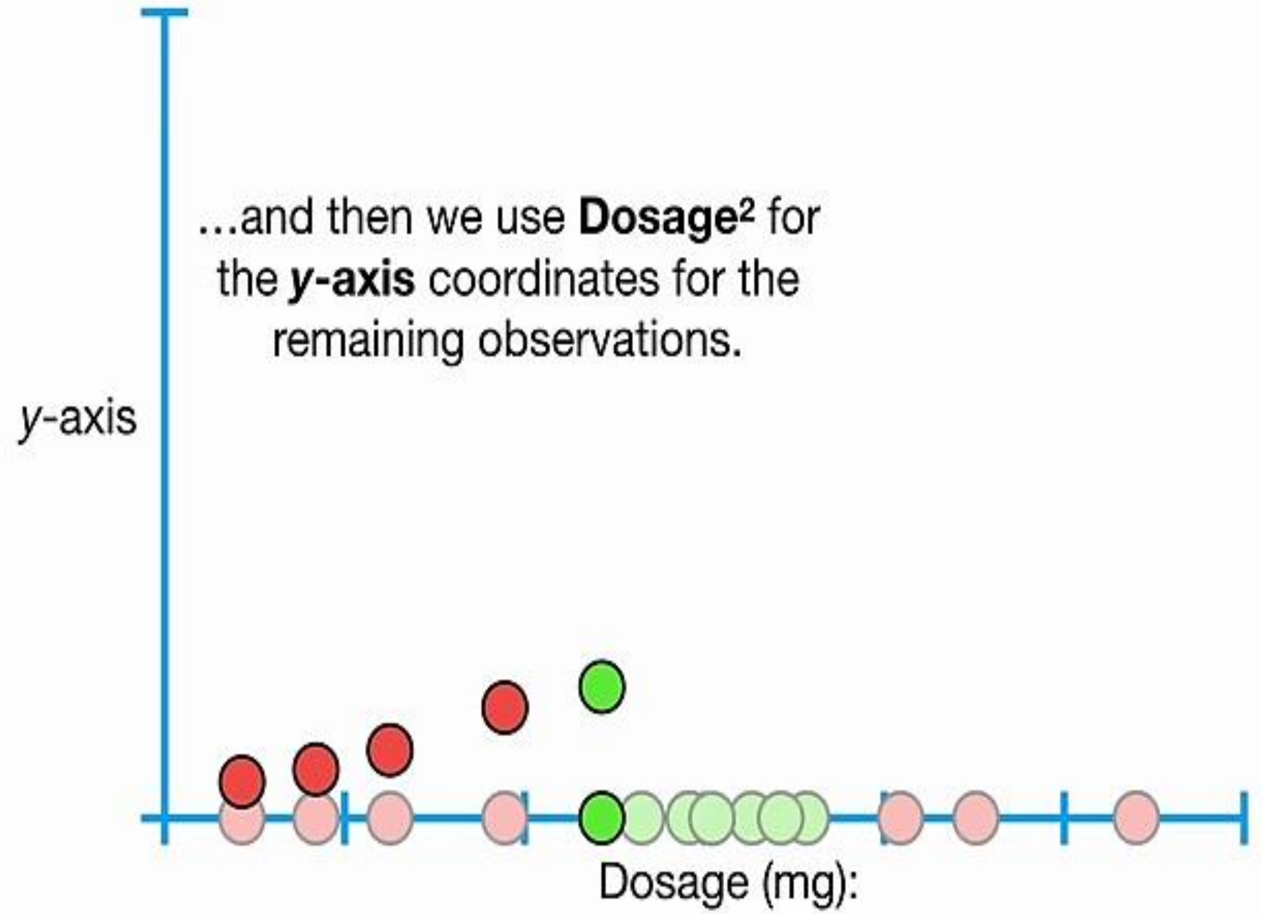


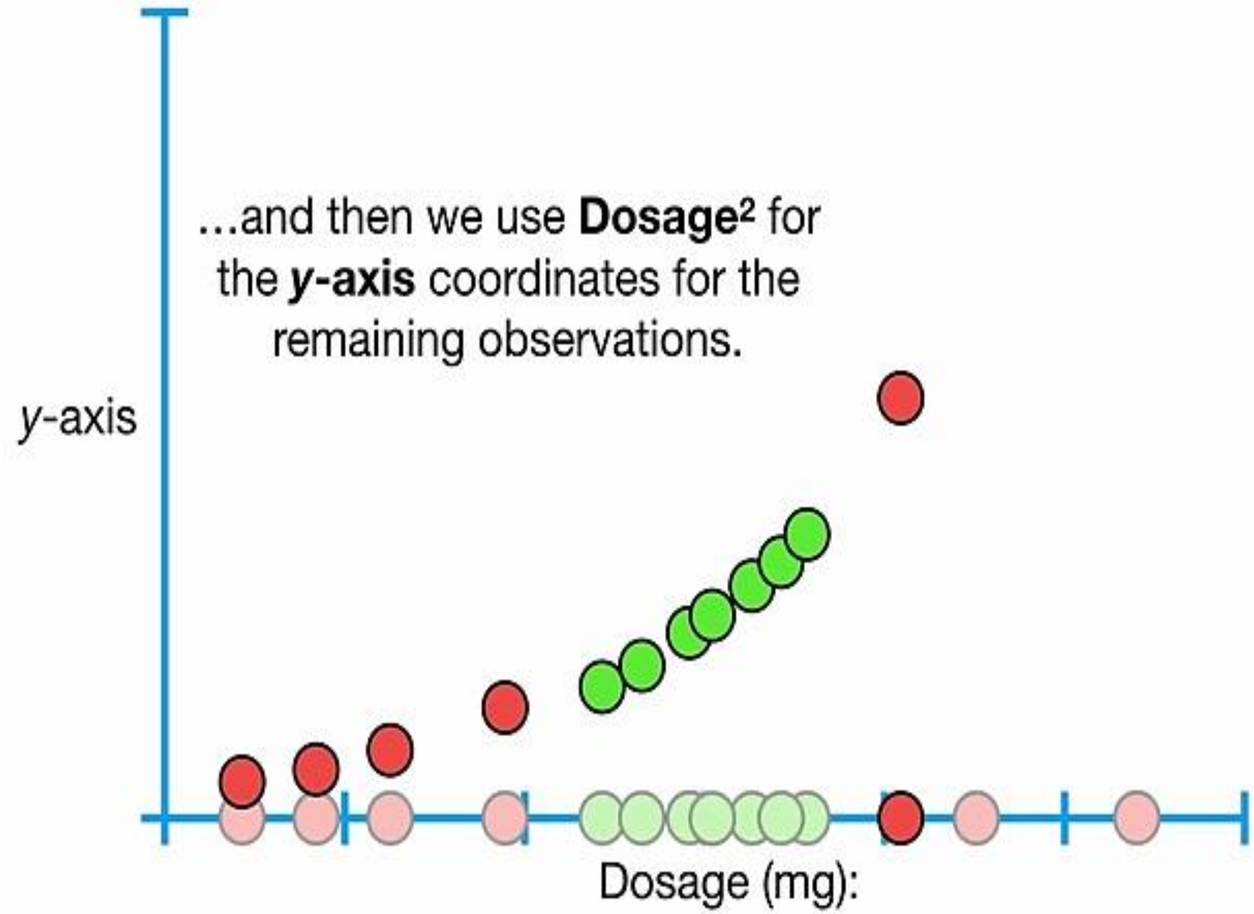
…the y-axis value = Dosage²
= 0.5²
= 0.25.
y-axis
0.25
Dosage (mg):

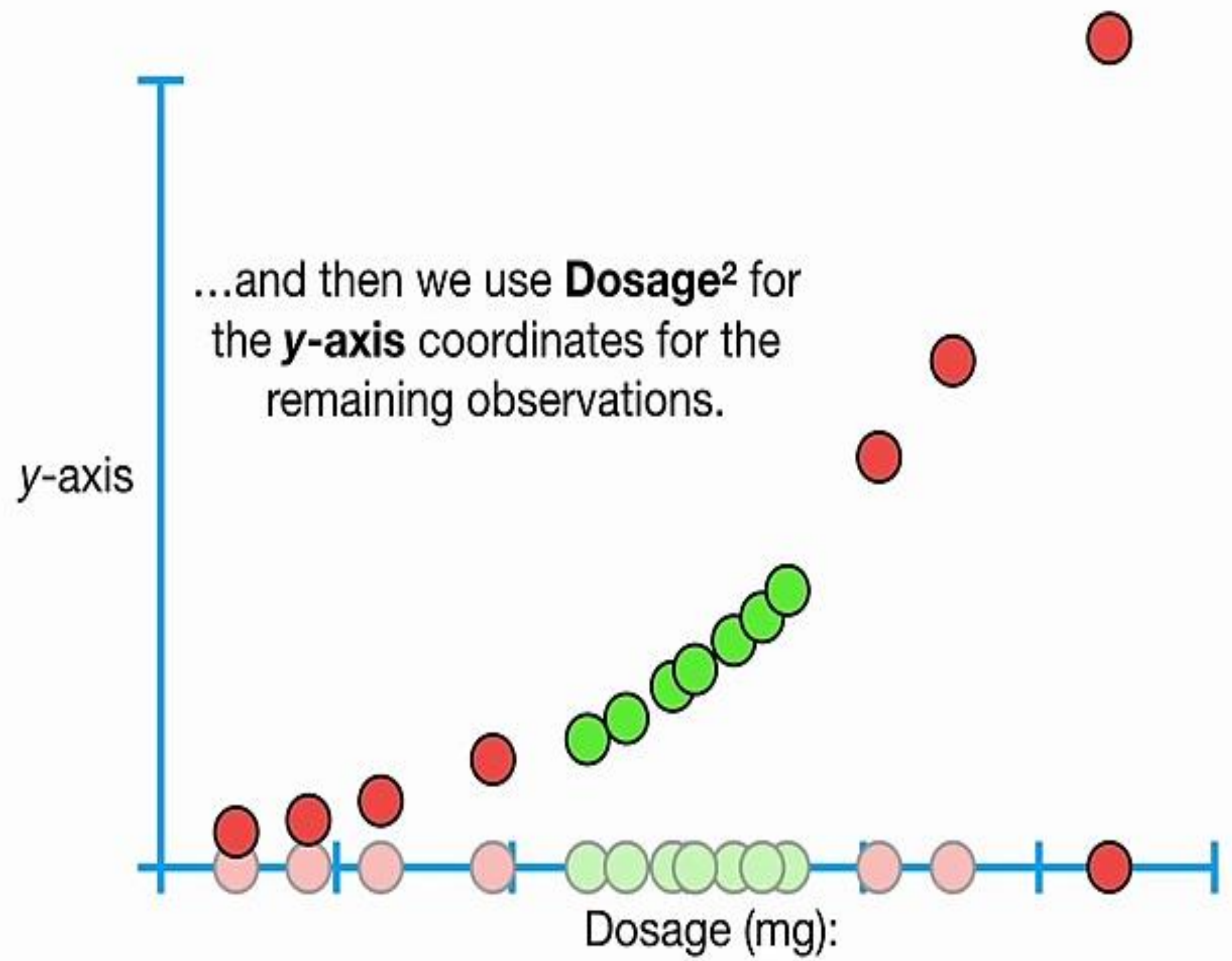

Now we use **Dosage²** for this ***y*-axis** coordinate…
y-axis
Dosage (mg):

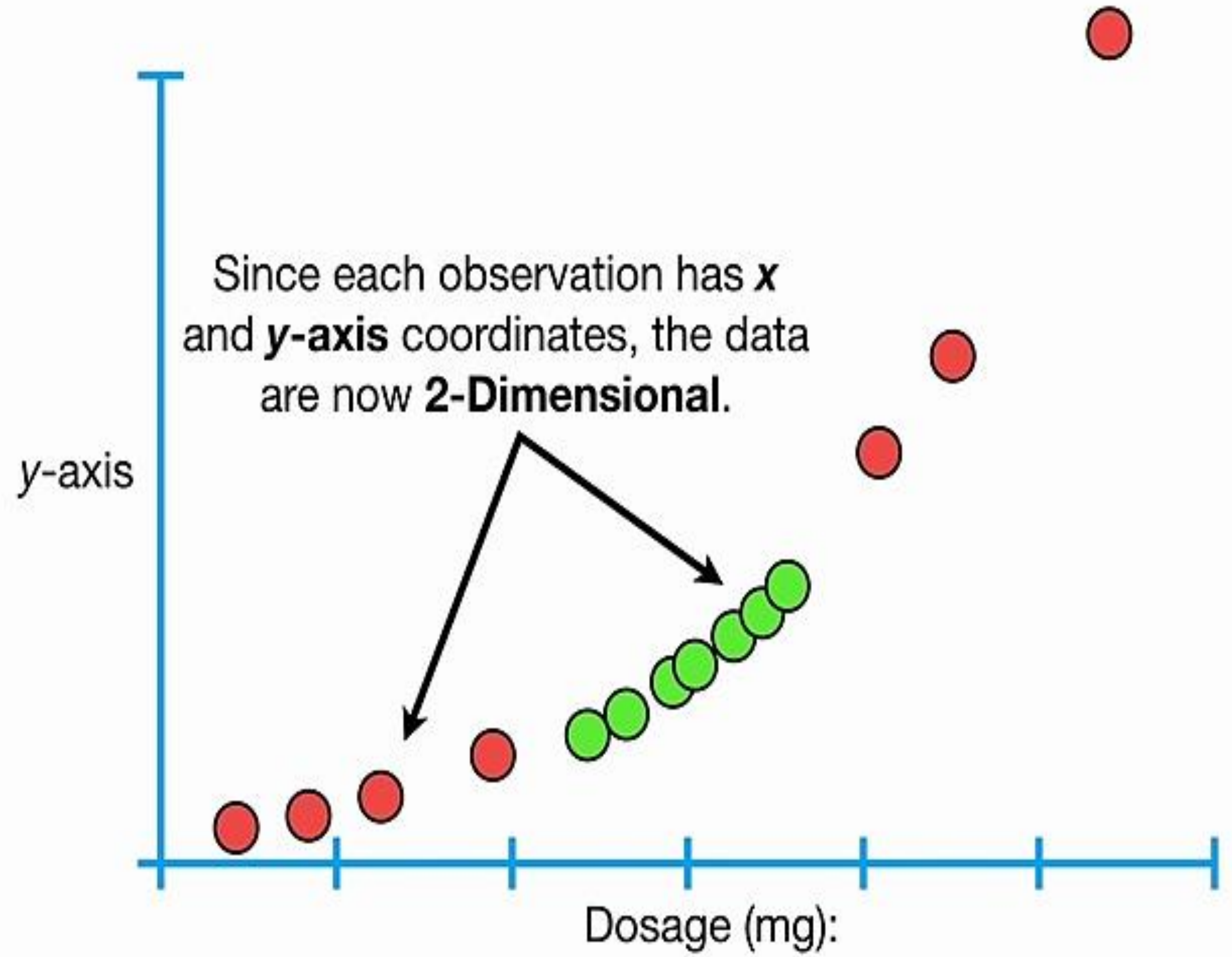

Now we use **Dosage²** for this ***y*-axis** coordinate…
y-axis
Dosage (mg):

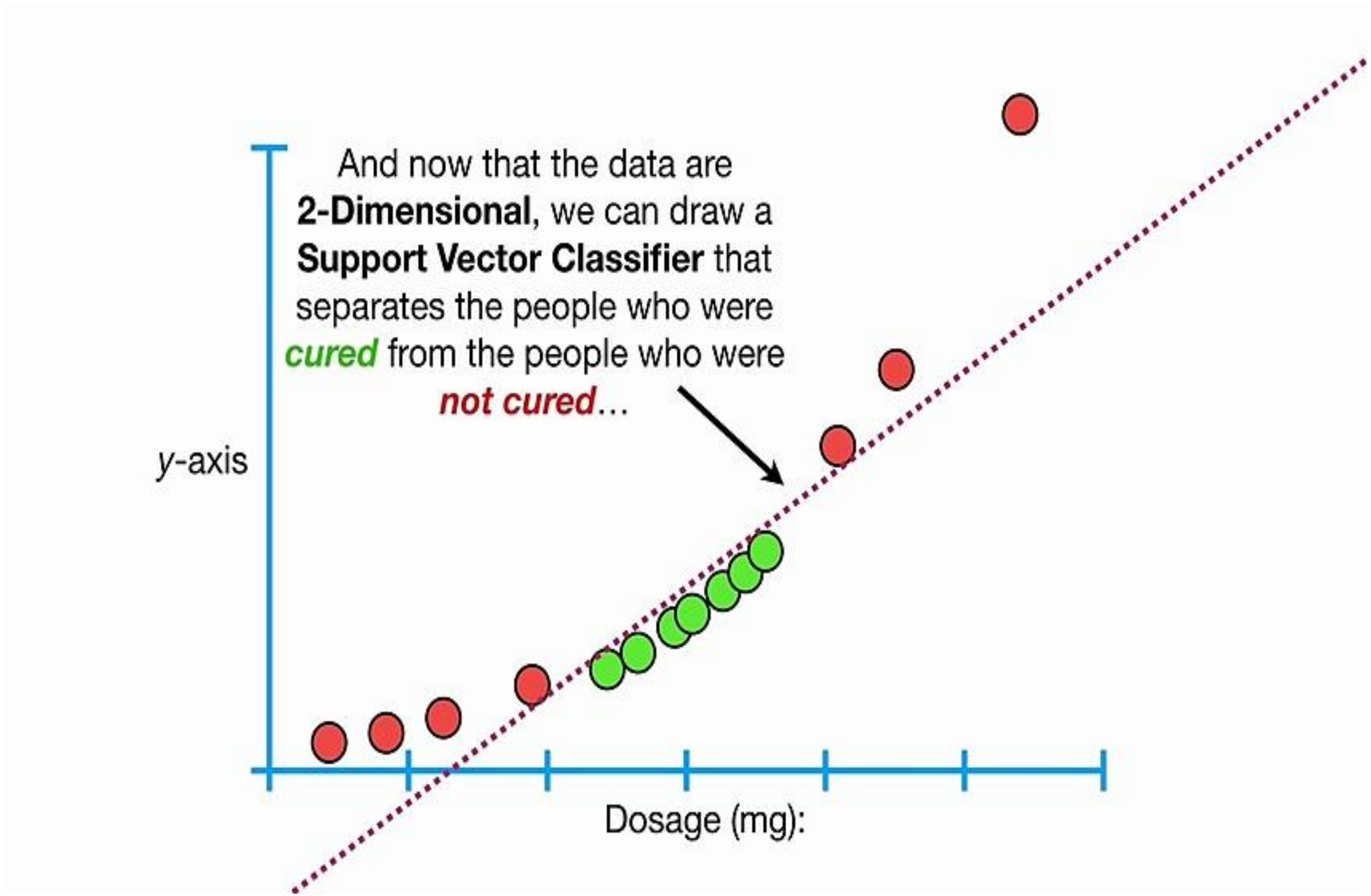

...and then we use **Dosage²** for the ***y*-axis** coordinates for the remaining observations.
y-axis
Dosage (mg):

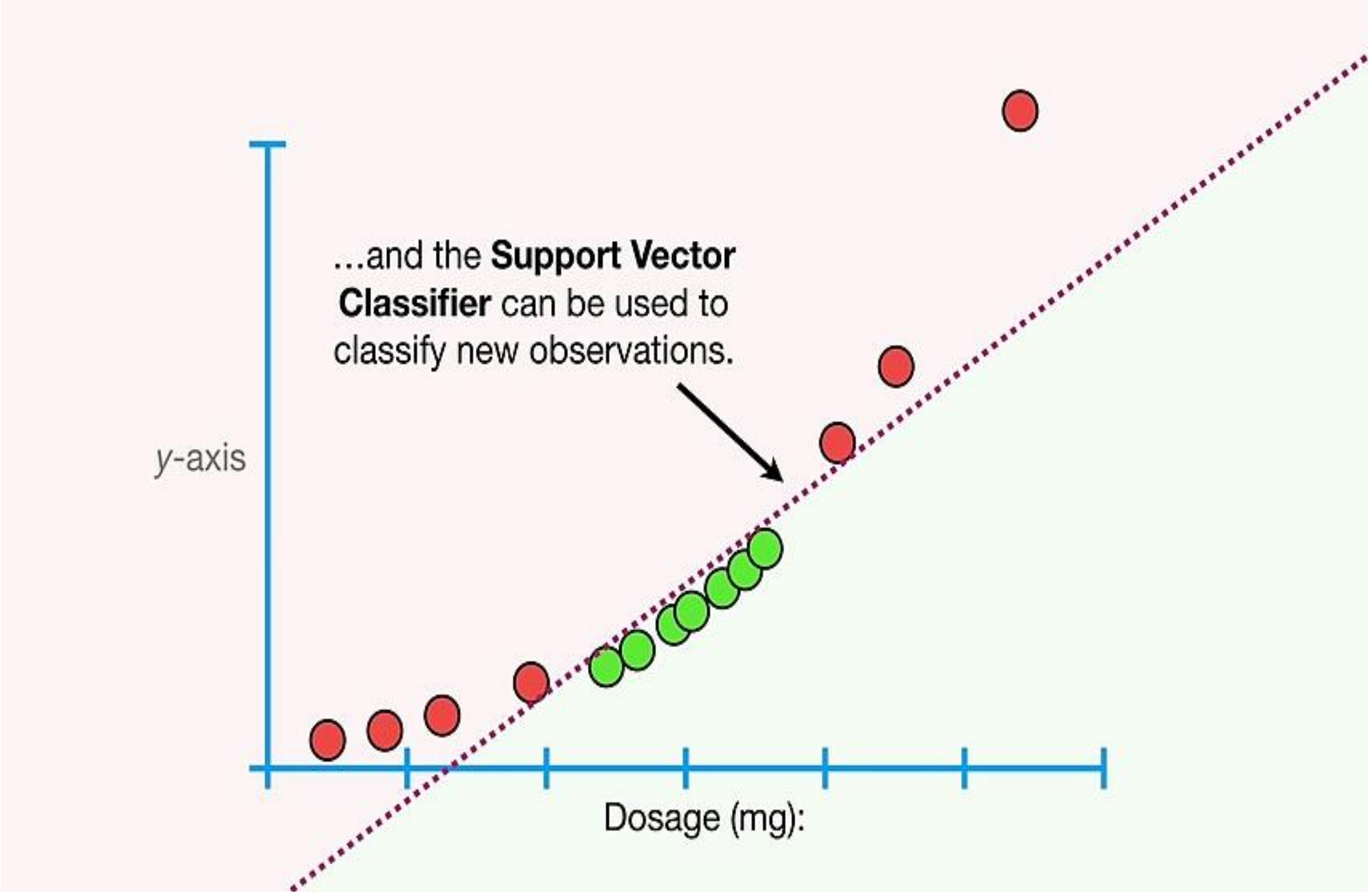

...and then we use Dosage² for the y-axis coordinates for the remaining observations.
y-axis
Dosage (mg):

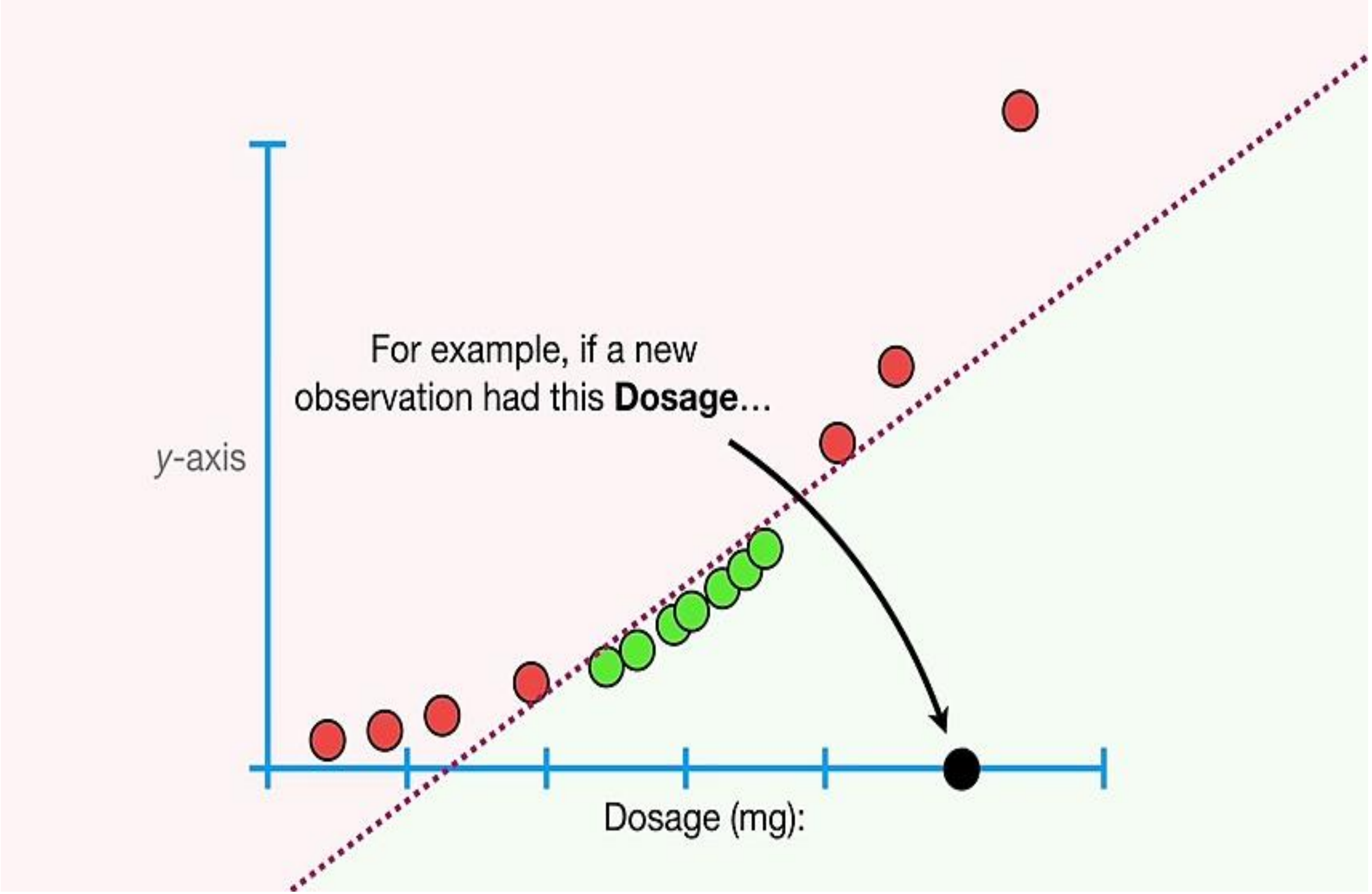

...and then we use **Dosage²** for the ***y*-axis** coordinates for the remaining observations.
y-axis
Dosage (mg):

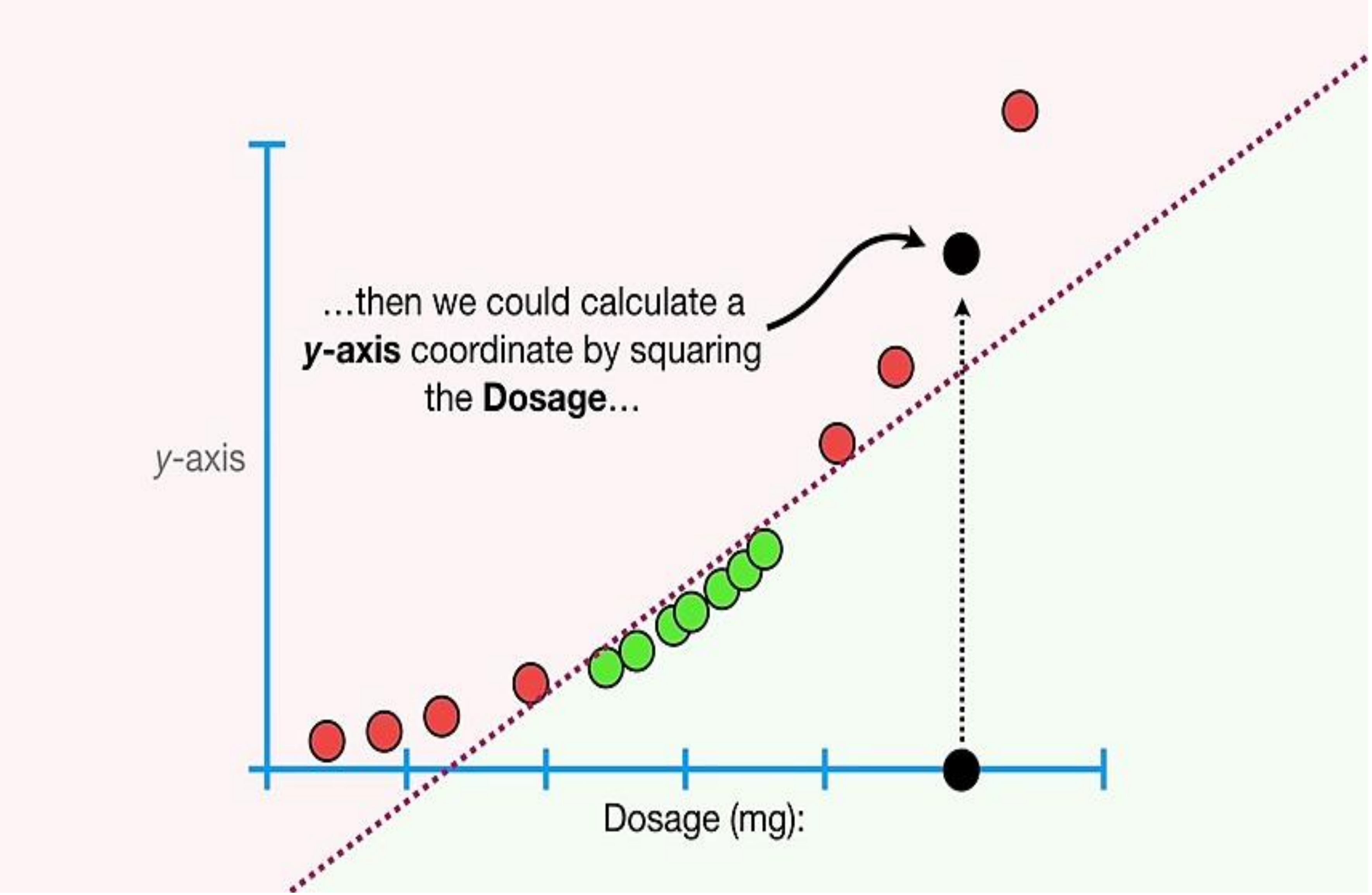

...and then we use **Dosage²** for
the ***y*-axis** coordinates for the
remaining observations.
y-axis
Dosage (mg):

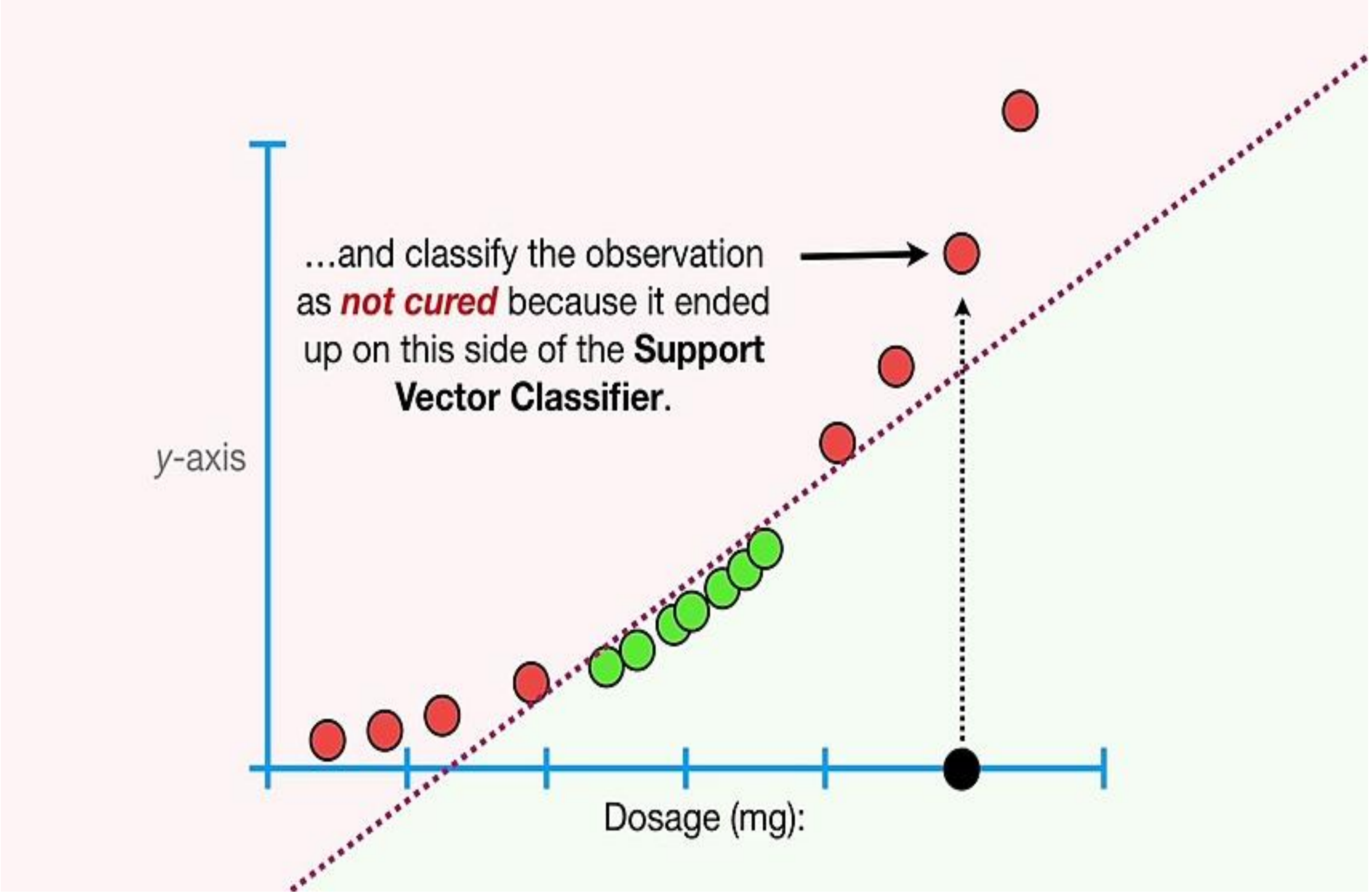

Since each observation has ***x***
and ***y*-axis** coordinates, the data
are now **2-Dimensional**.
y-axis
Dosage (mg):

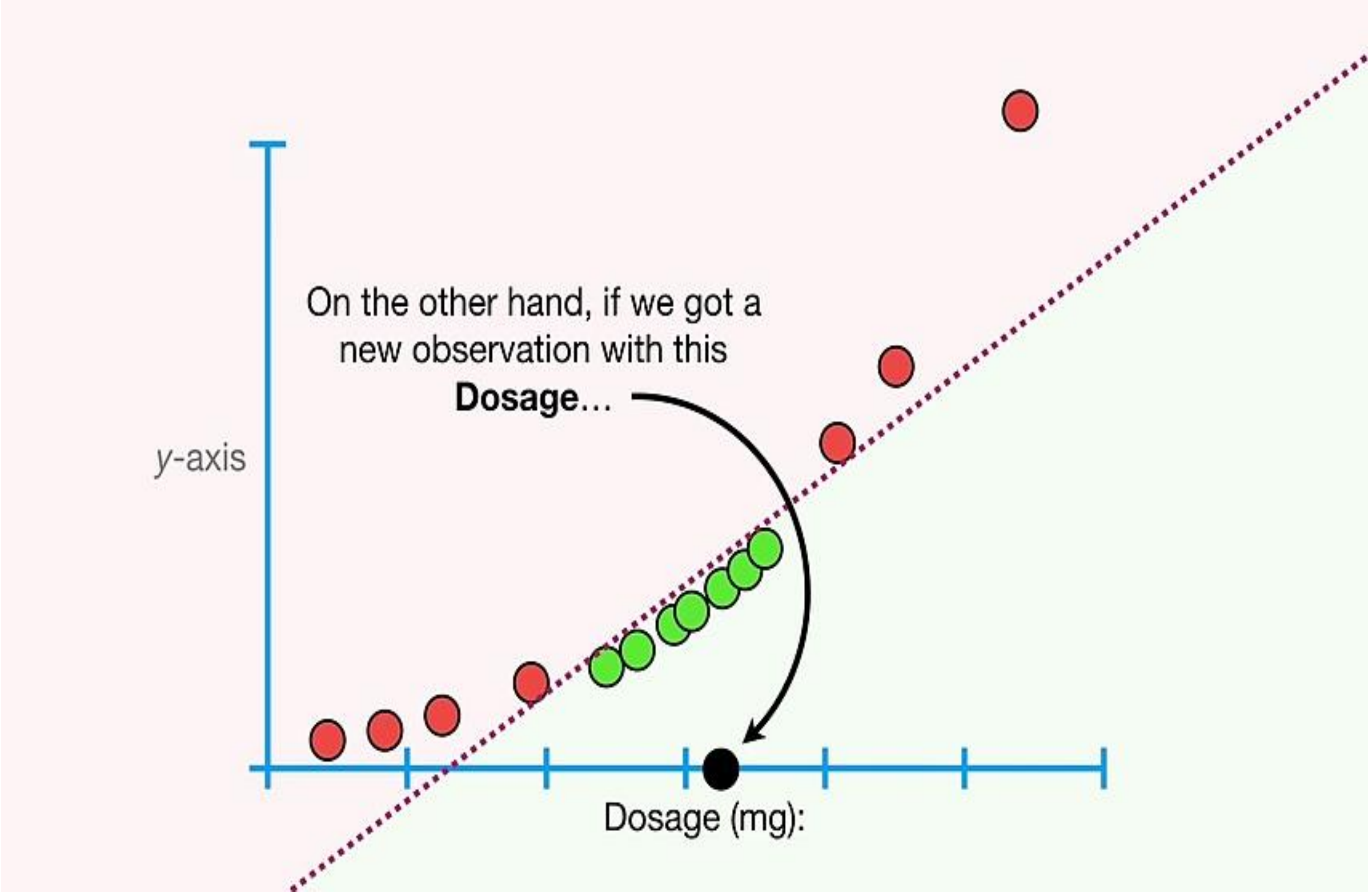

And now that the data are
2-Dimensional, we can draw a
Support Vector Classifier that
separates the people who were
cured from the people who were
not cured…
y-axis
Dosage (mg):

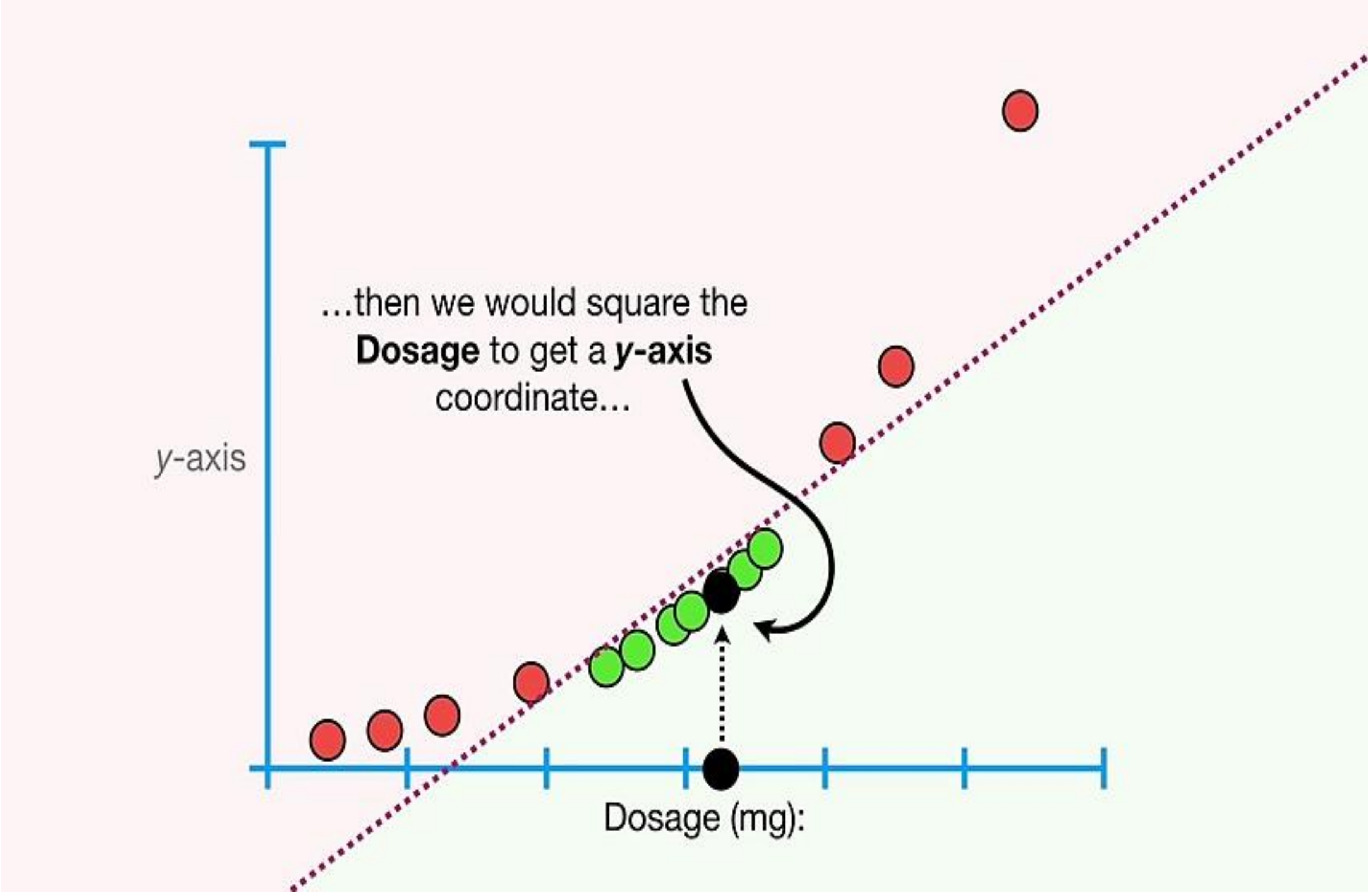

...and the **Support Vector Classifier** can be used to classify new observations.
y-axis
Dosage (mg):

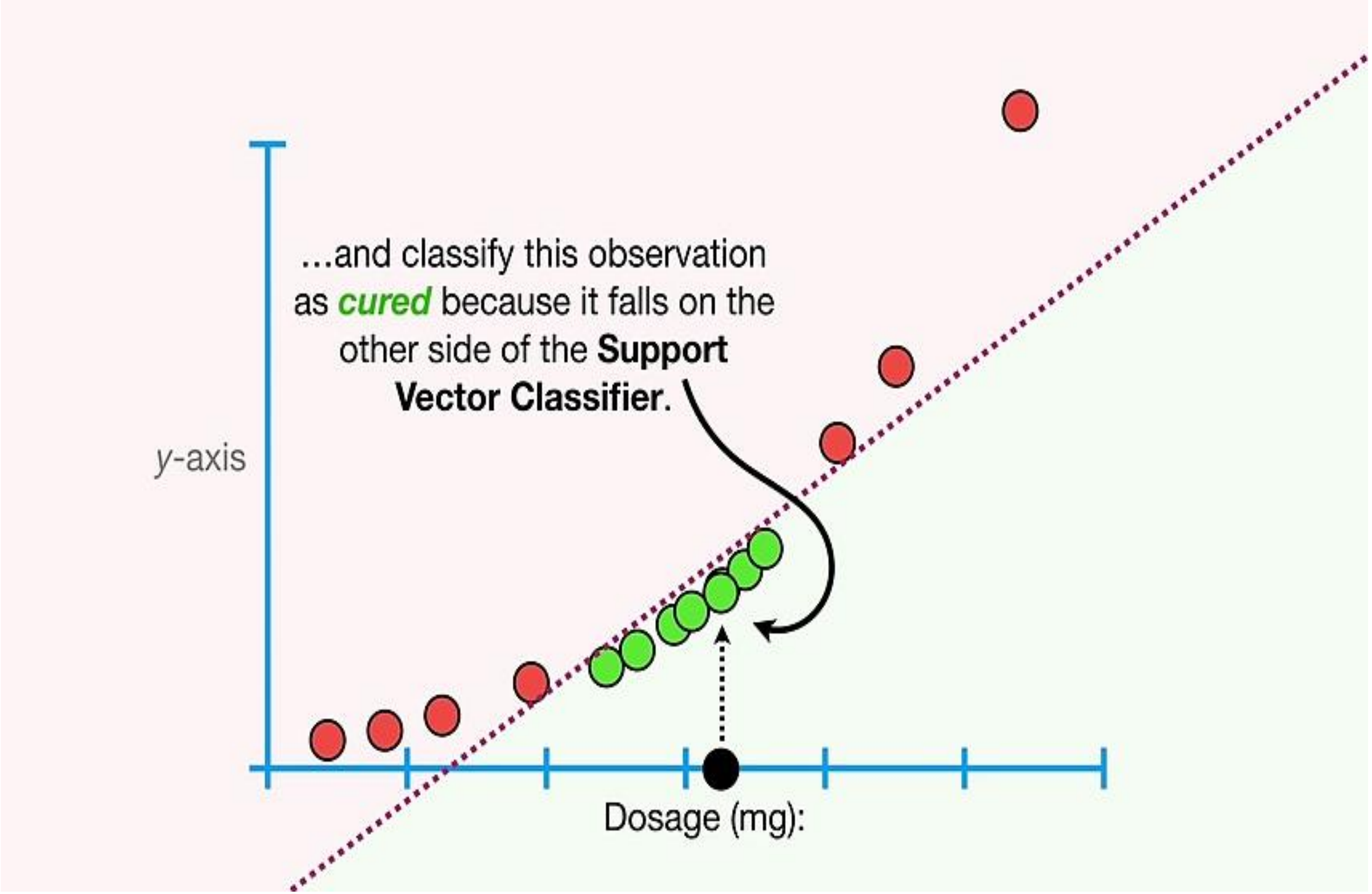

For example, if a new observation had this **Dosage**...
y-axis
Dosage (mg):


...then we could calculate a
y-axis coordinate by squaring
the Dosage...
y-axis
Dosage (mg):

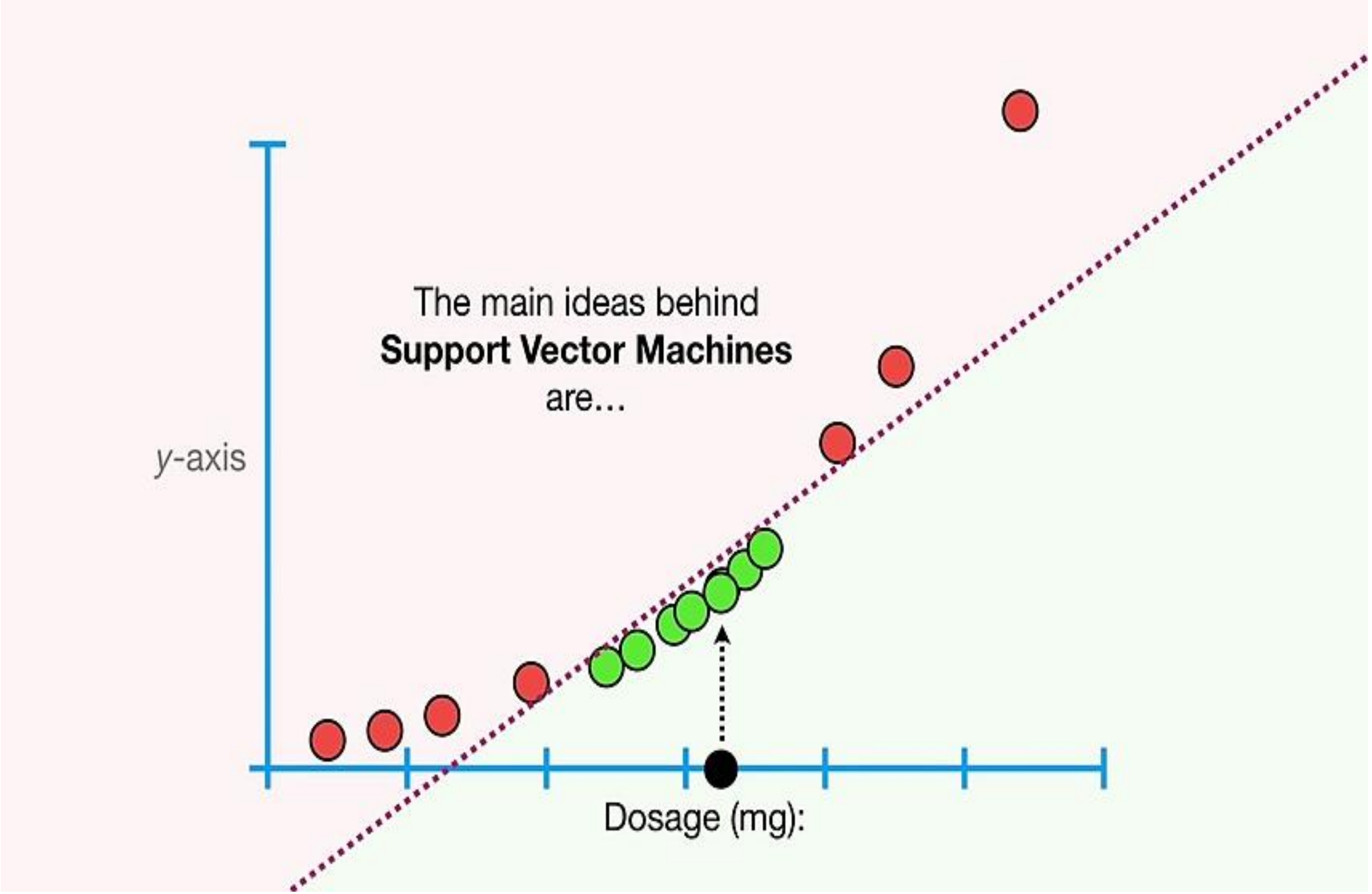

...and classify the observation as *not cured* because it ended up on this side of the **Support Vector Classifier**.
y-axis
Dosage (mg):


On the other hand, if we got a new observation with this **Dosage**…
y-axis
Dosage (mg):

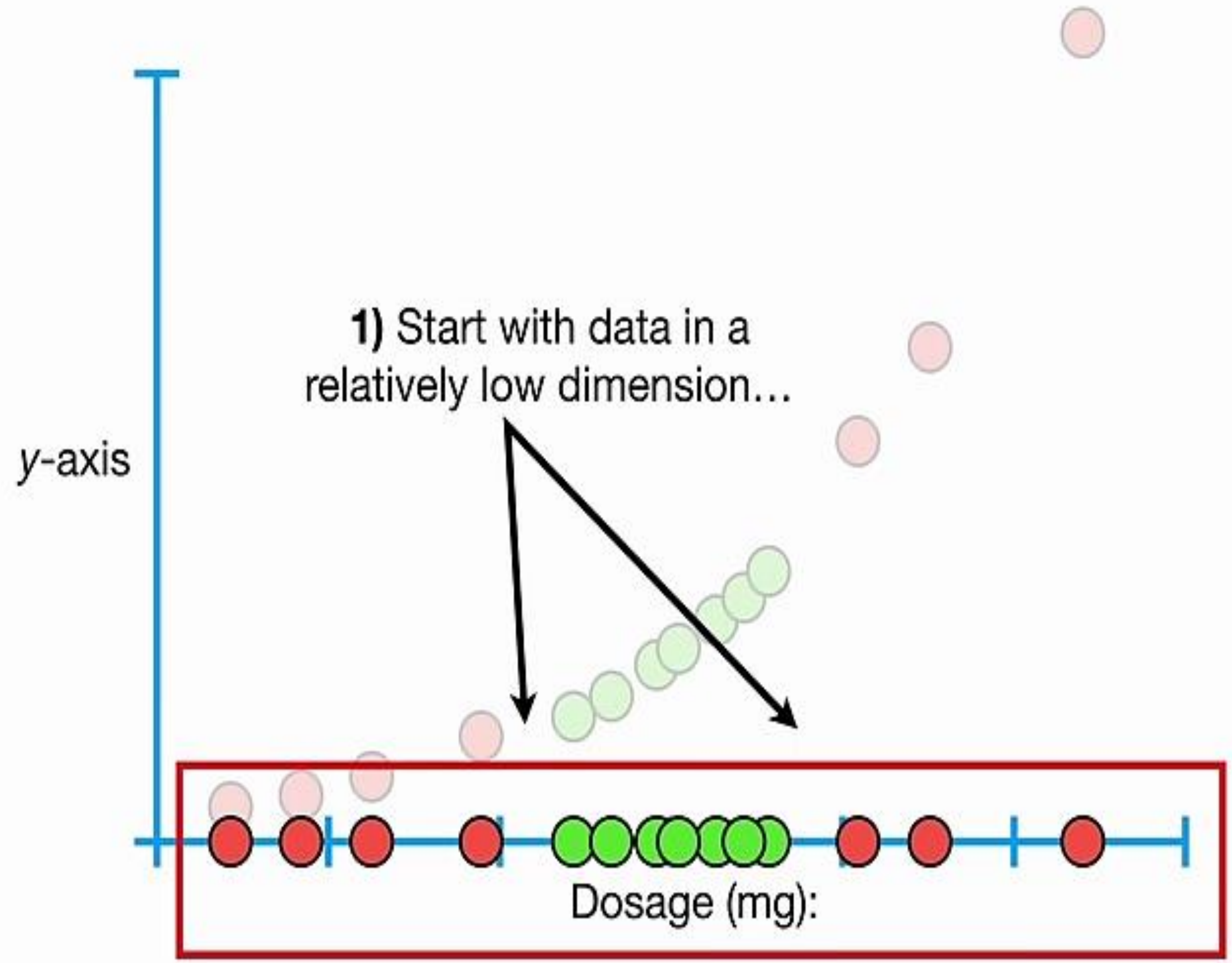

...then we would square the
Dosage to get a ***y*-axis**
coordinate...
y-axis
Dosage (mg):



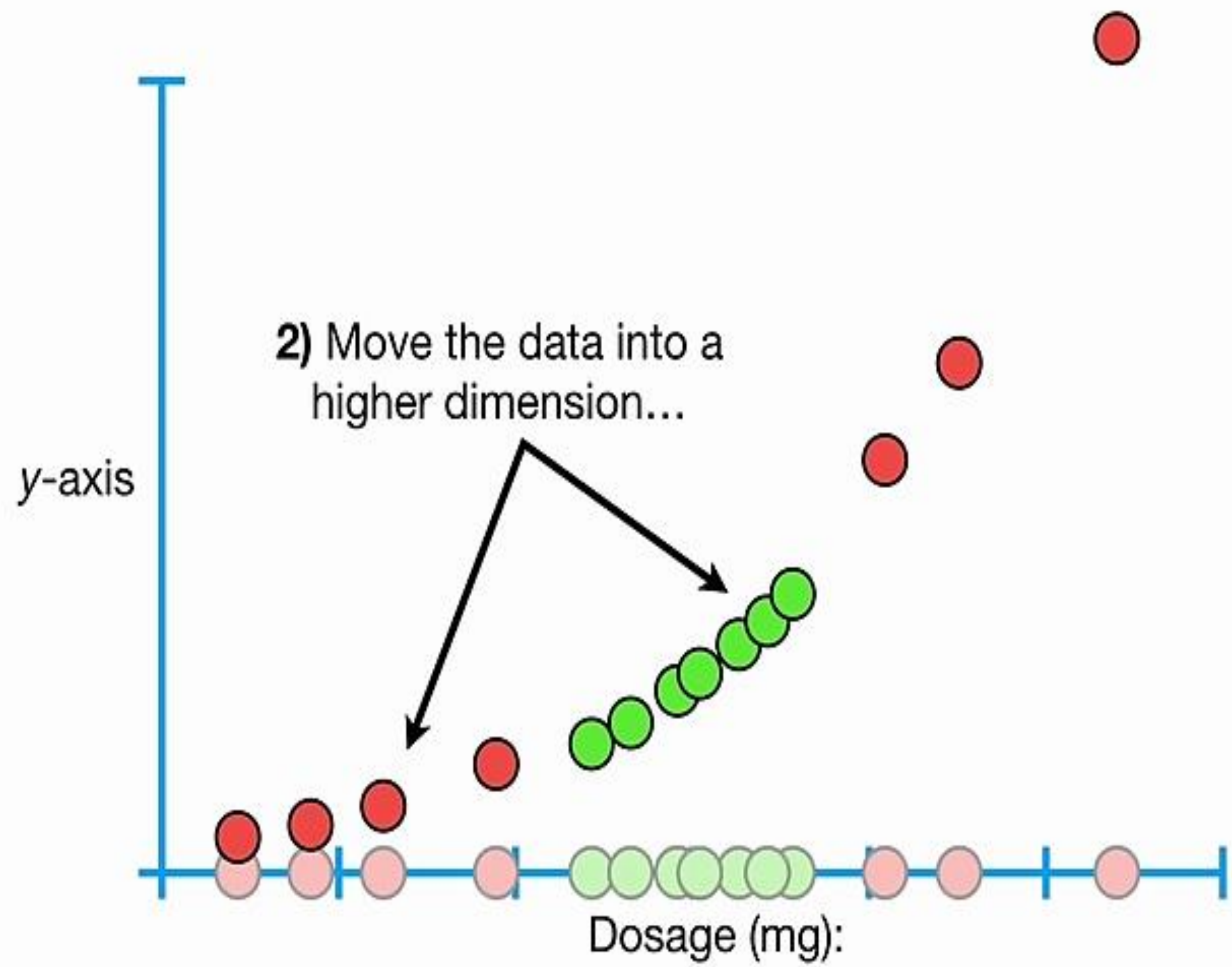
...and classify this observation as cured because it falls on the other side of the Support Vector Classifier.
y-axis
Dosage (mg):



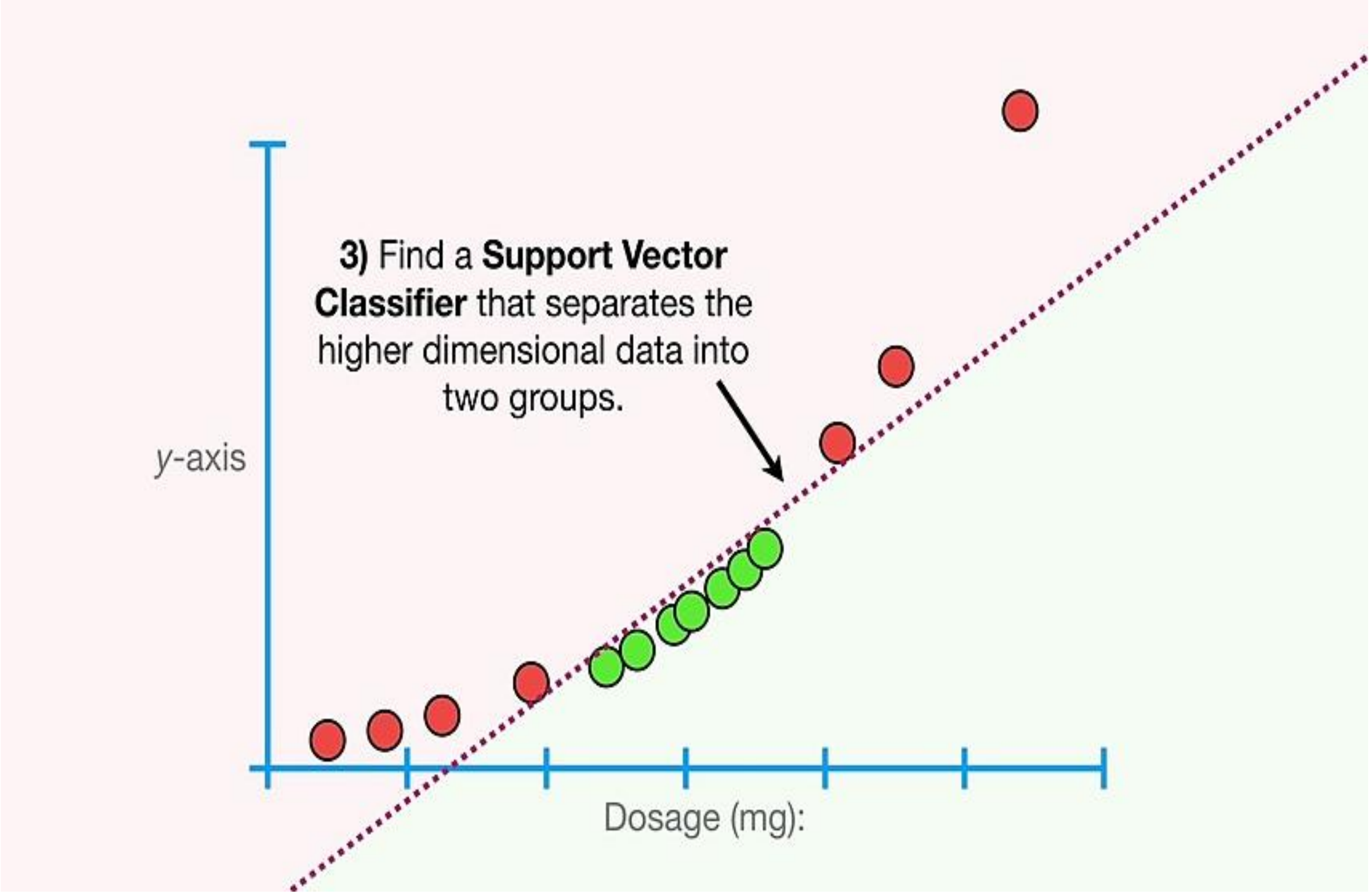
The main ideas behind
Support Vector Machines
are…
y-axis
Dosage (mg):

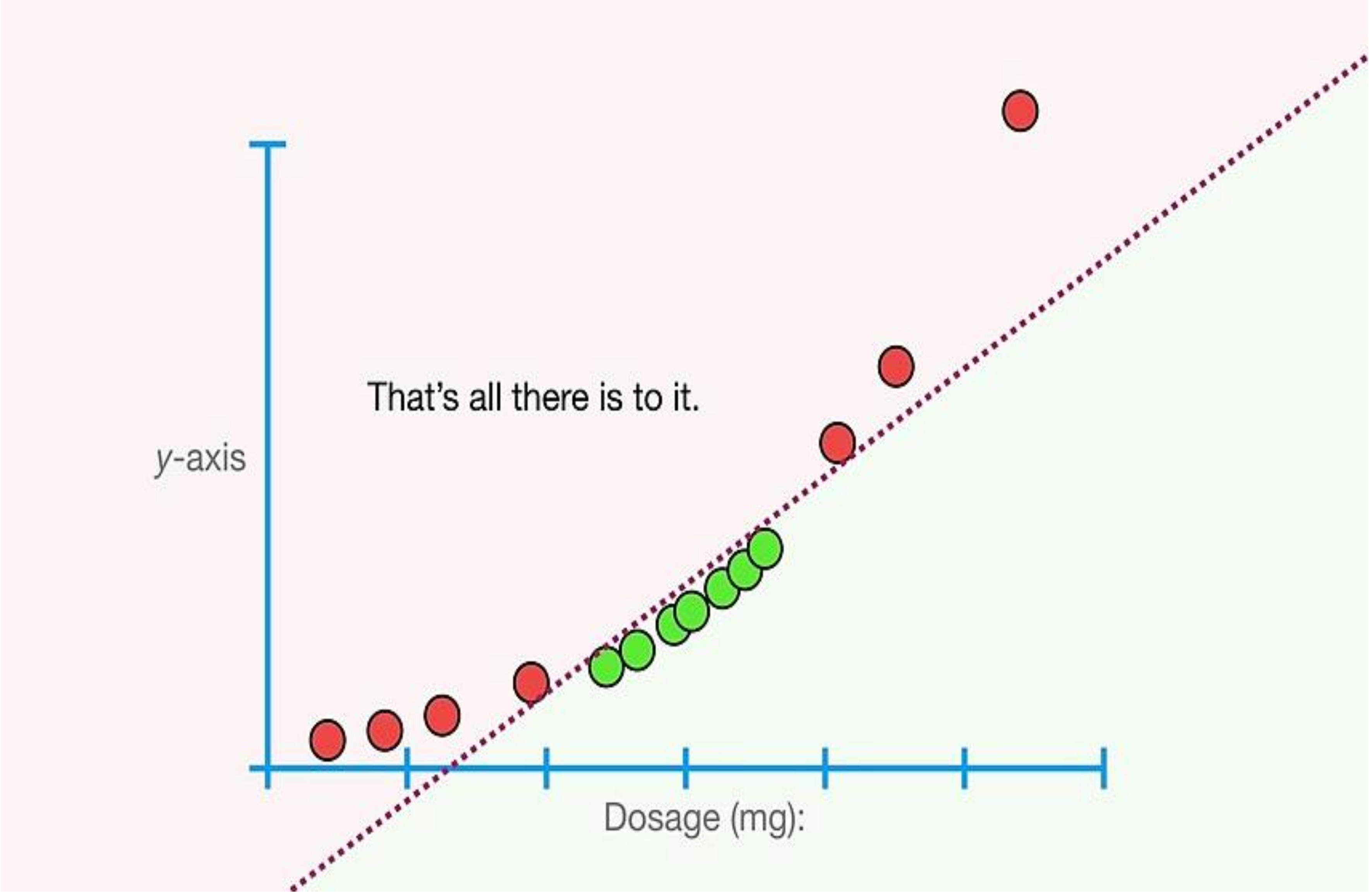


1) Start with data in a relatively low dimension...
y-axis
Dosage (mg):

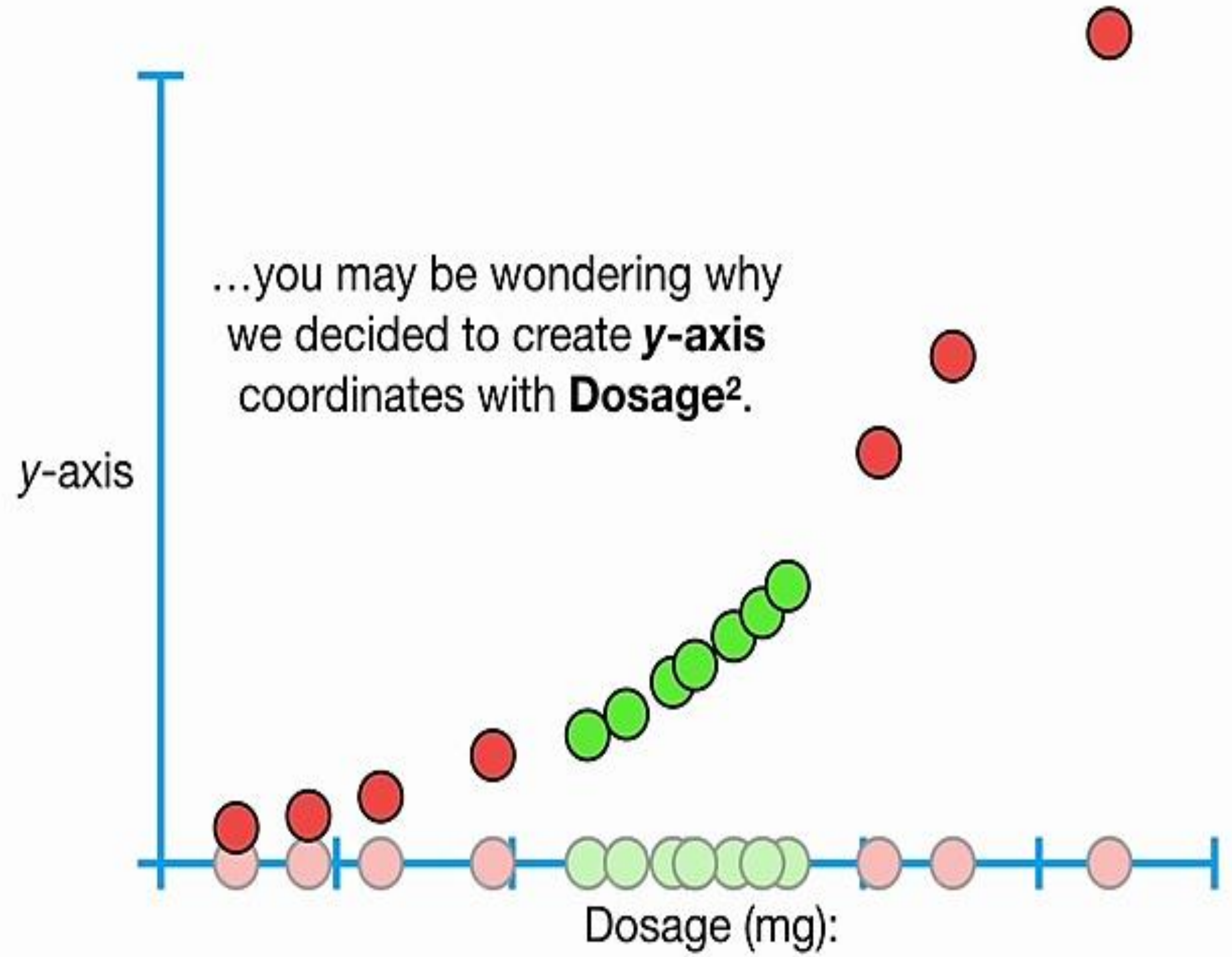


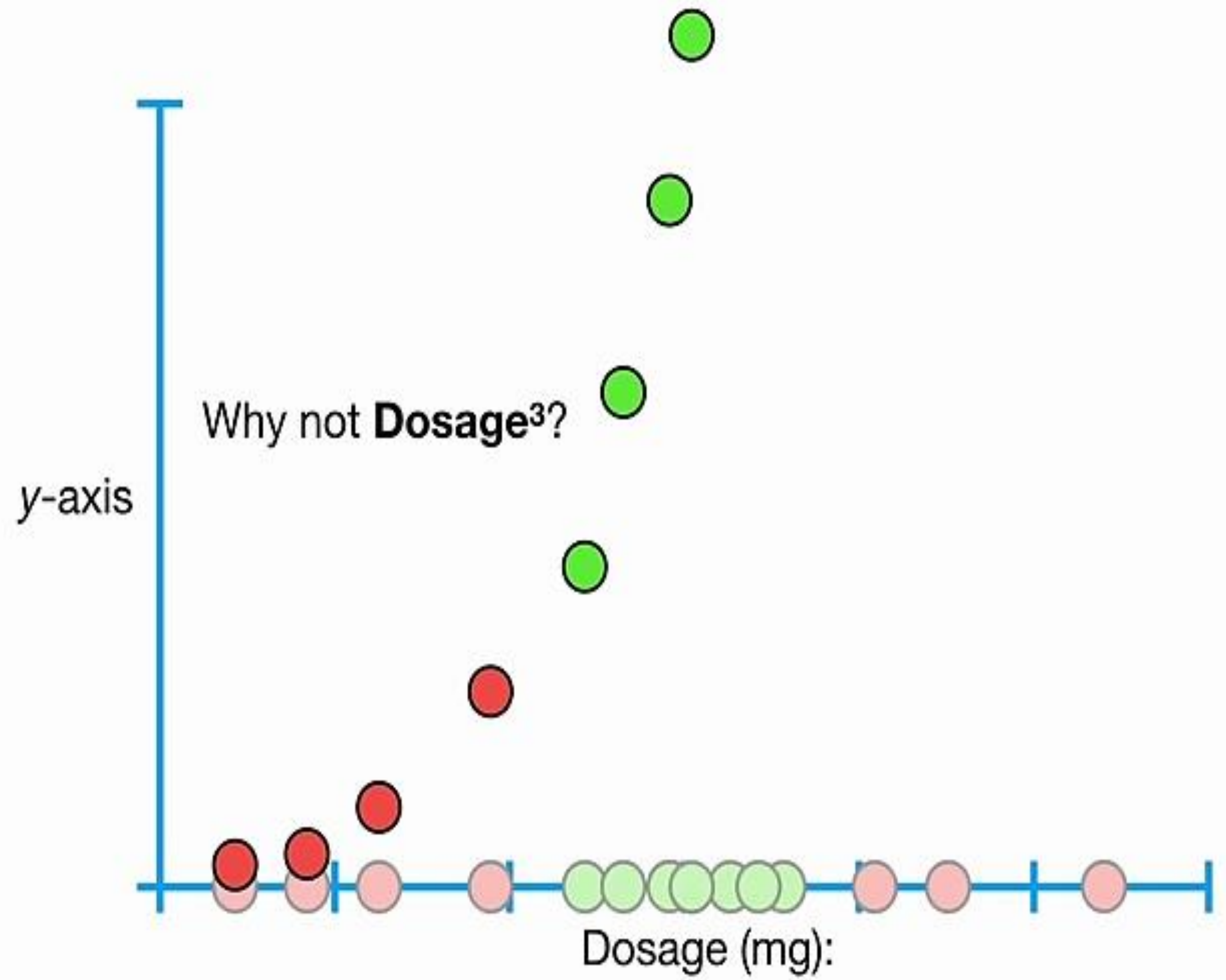
2) Move the data into a higher dimension...
y-axis
Dosage (mg):

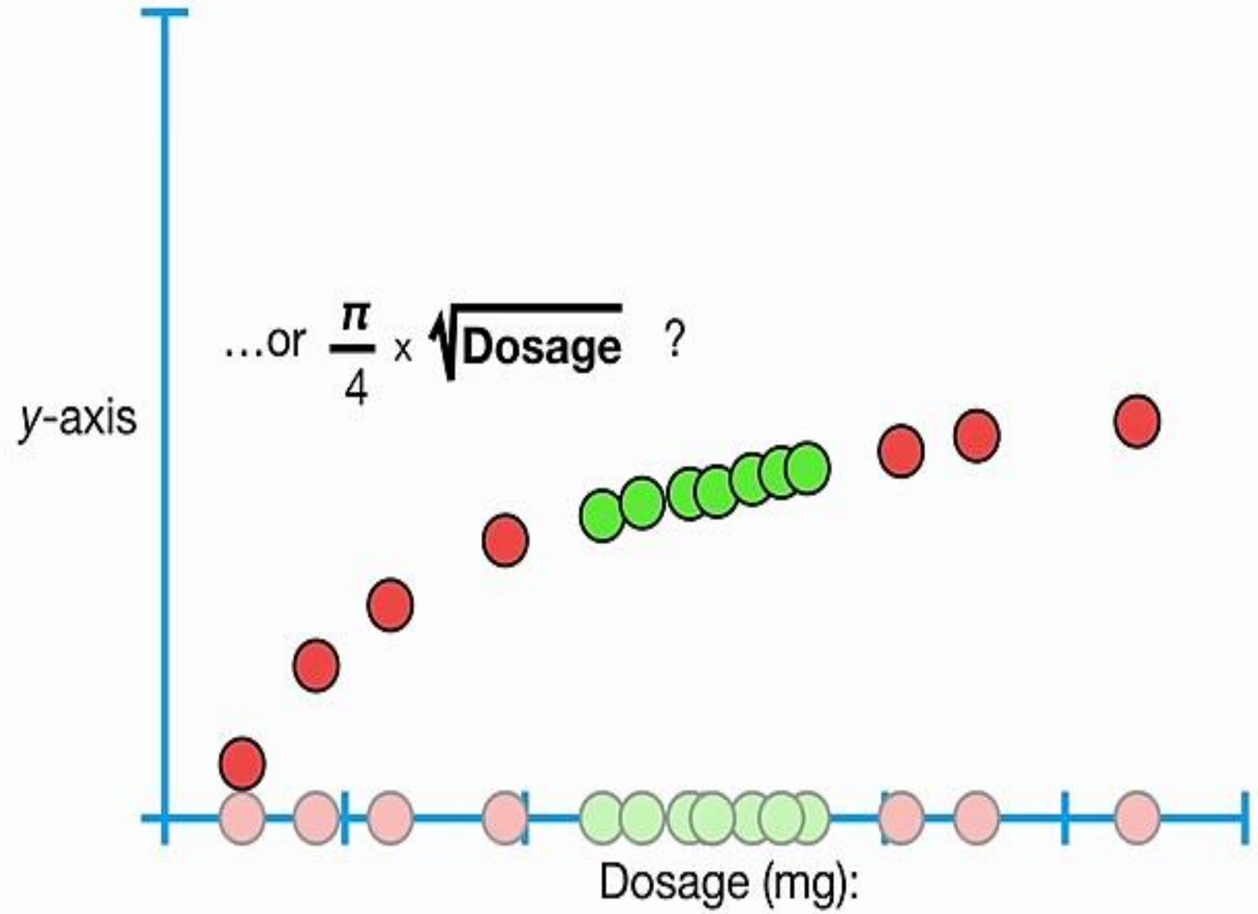

3) Find a **Support Vector Classifier** that separates the higher dimensional data into two groups.
y-axis
Dosage (mg):

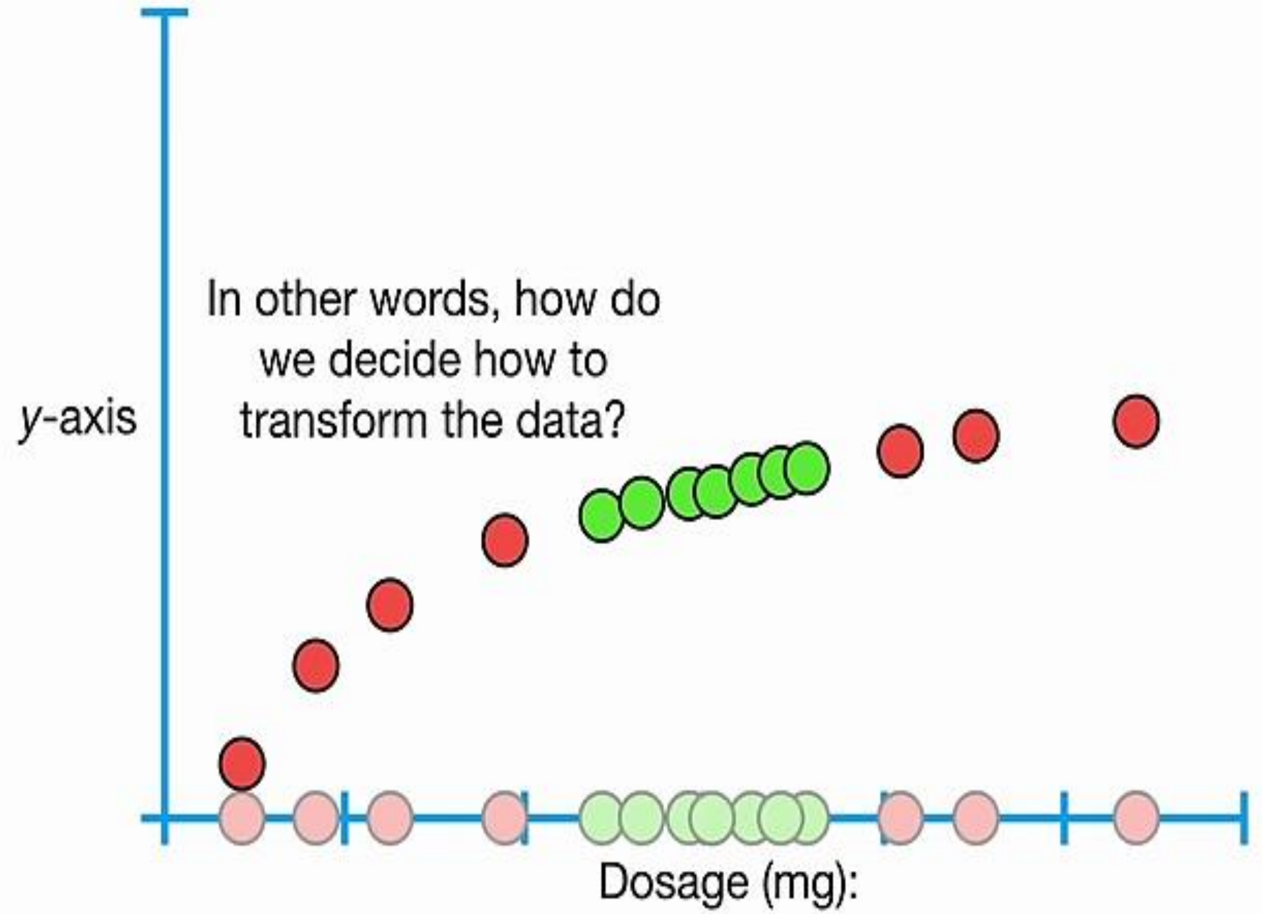

That's all there is to it.
y-axis
Dosage (mg):

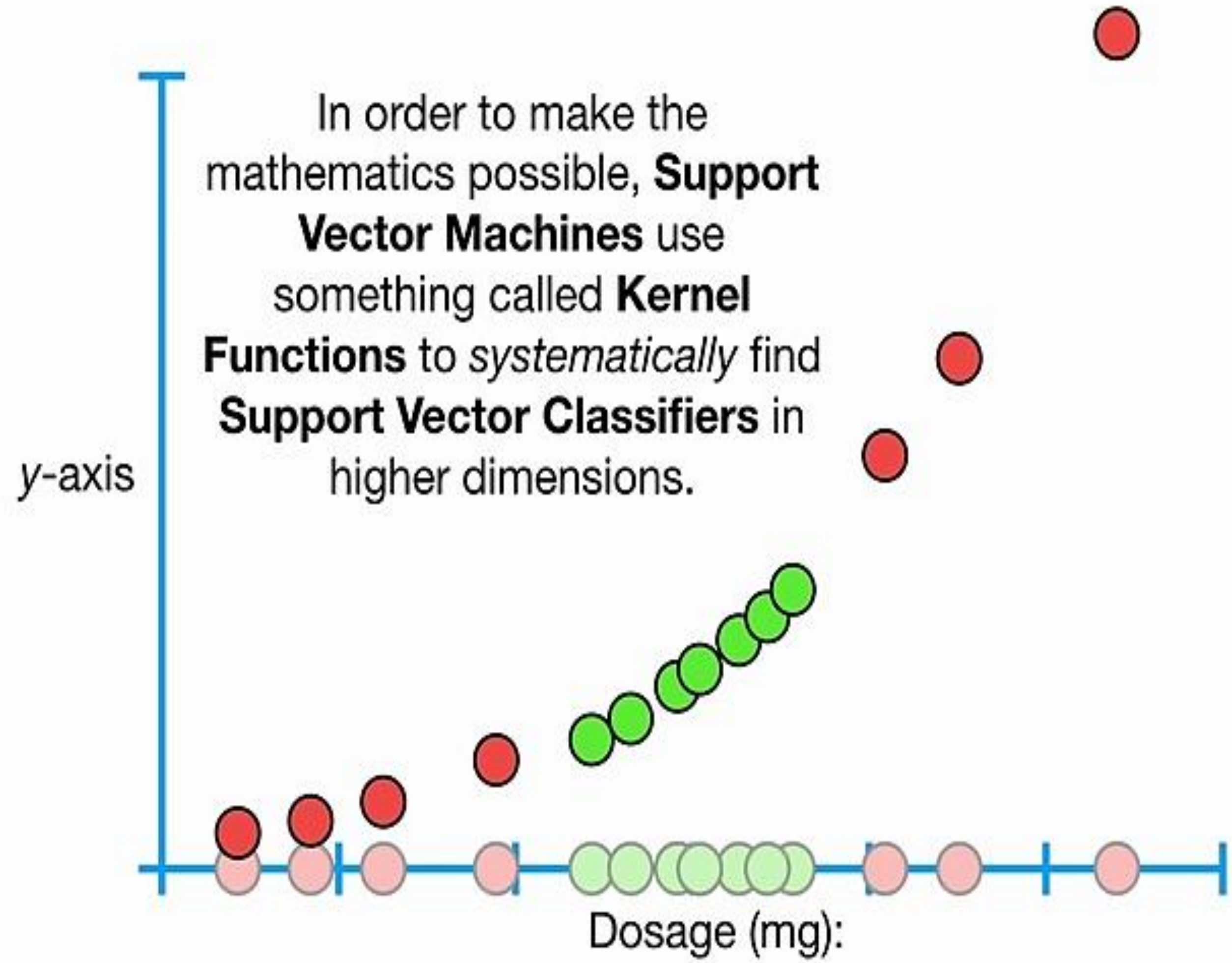
Going back to the original
1-Dimensional data…

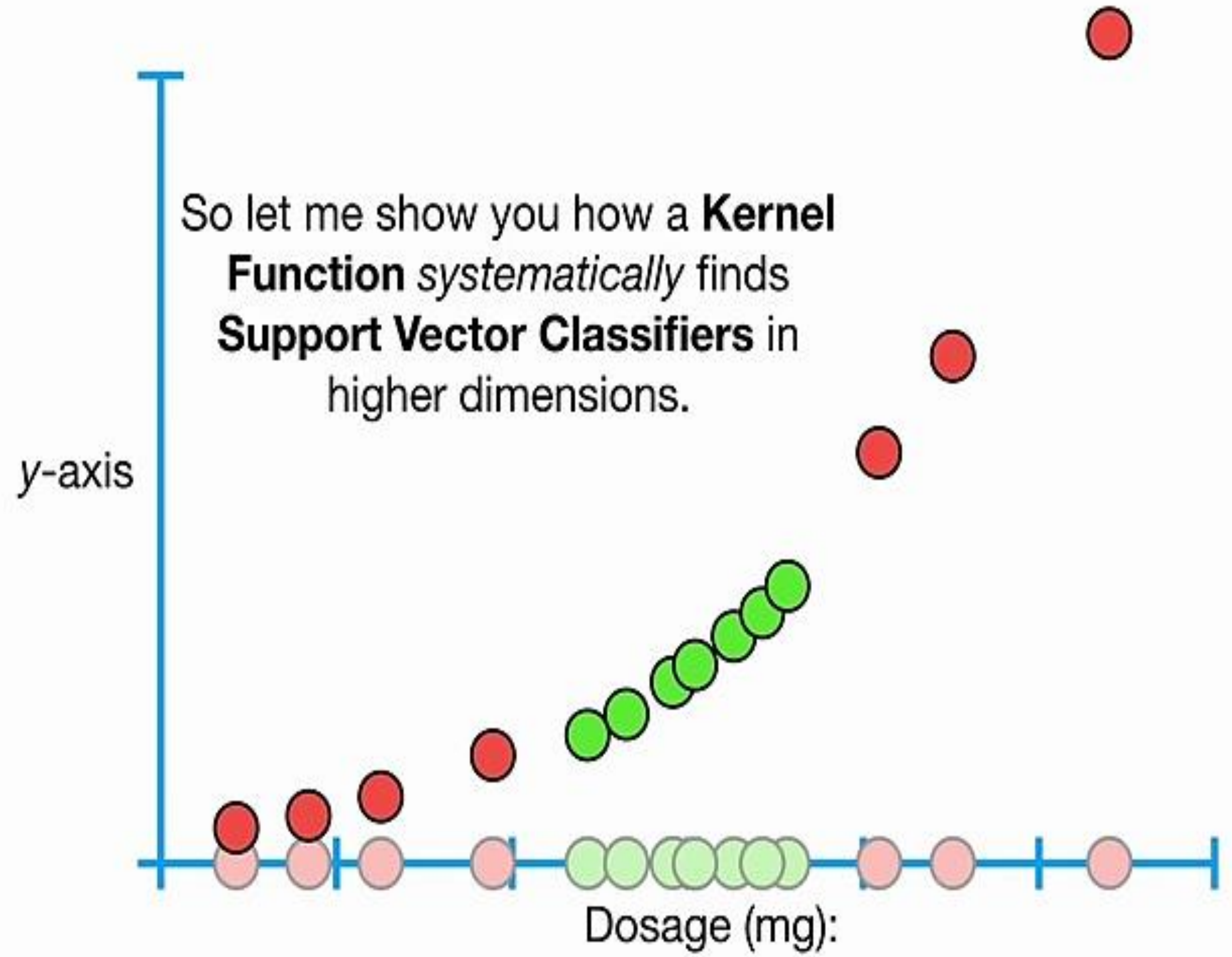

...you may be wondering why
we decided to create ***y*-axis**
coordinates with **Dosage²**.
y-axis
Dosage (mg):

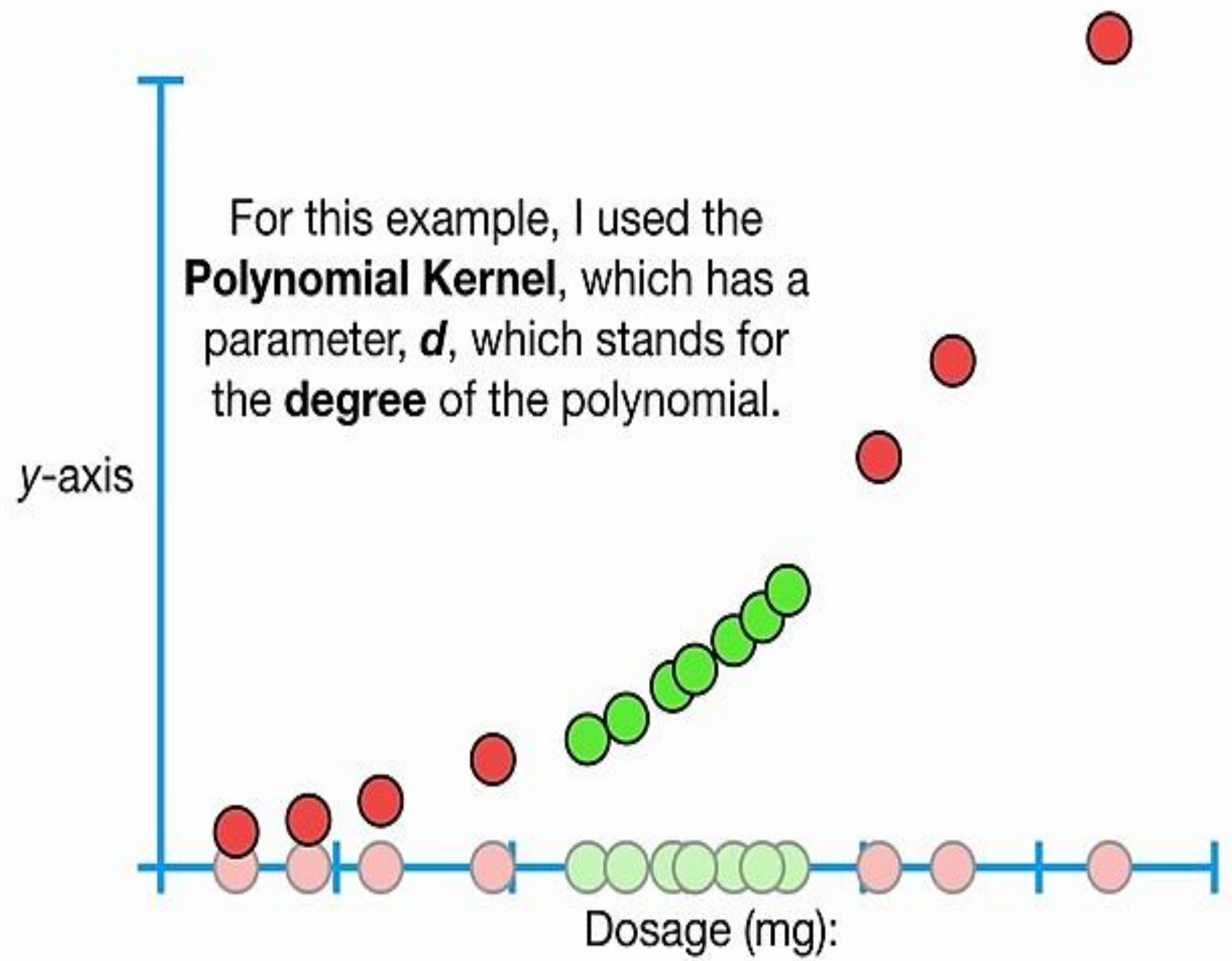

Why not Dosage³?
y-axis
Dosage (mg):


...or $\frac{\pi}{4}$ x $\sqrt{\textbf{Dosage}}$?
y-axis
Dosage (mg):


In other words, how do we decide how to transform the data?
y-axis
Dosage (mg):

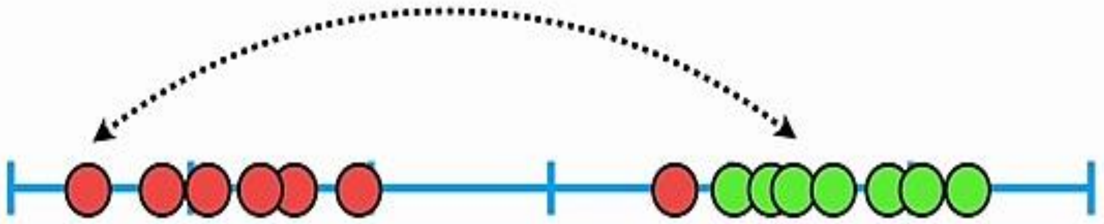

In order to make the mathematics possible, **Support Vector Machines** use something called **Kernel Functions** to *systematically* find **Support Vector Classifiers** in higher dimensions.
y-axis
Dosage (mg):


So let me show you how a **Kernel Function** *systematically* finds **Support Vector Classifiers** in higher dimensions.
y-axis
Dosage (mg):

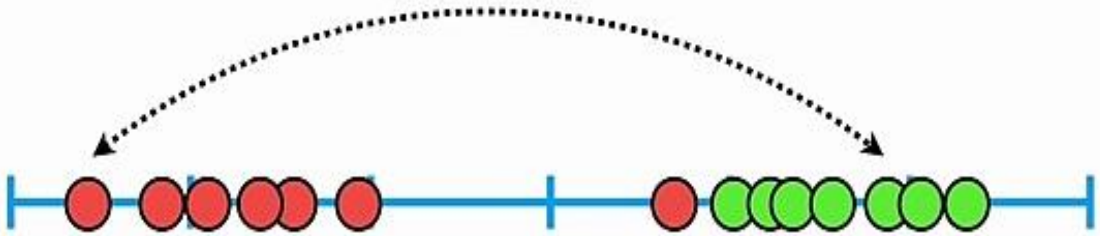

For this example, I used the **Polynomial Kernel**, which has a parameter, ***d***, which stands for the **degree** of the polynomial.
y-axis
Dosage (mg):

When ***d* = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…

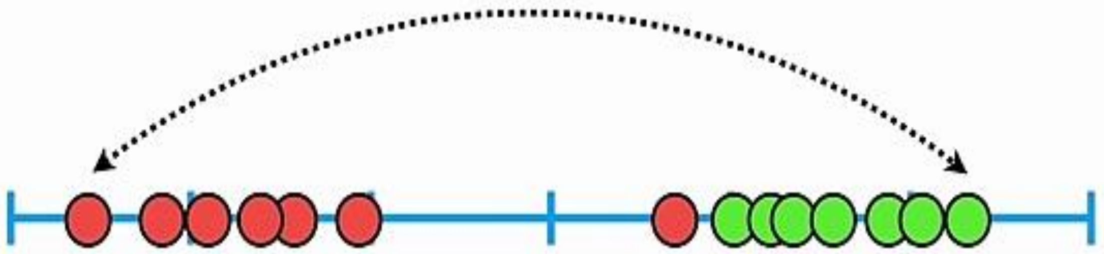
When ***d* = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…



When ***d* = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…

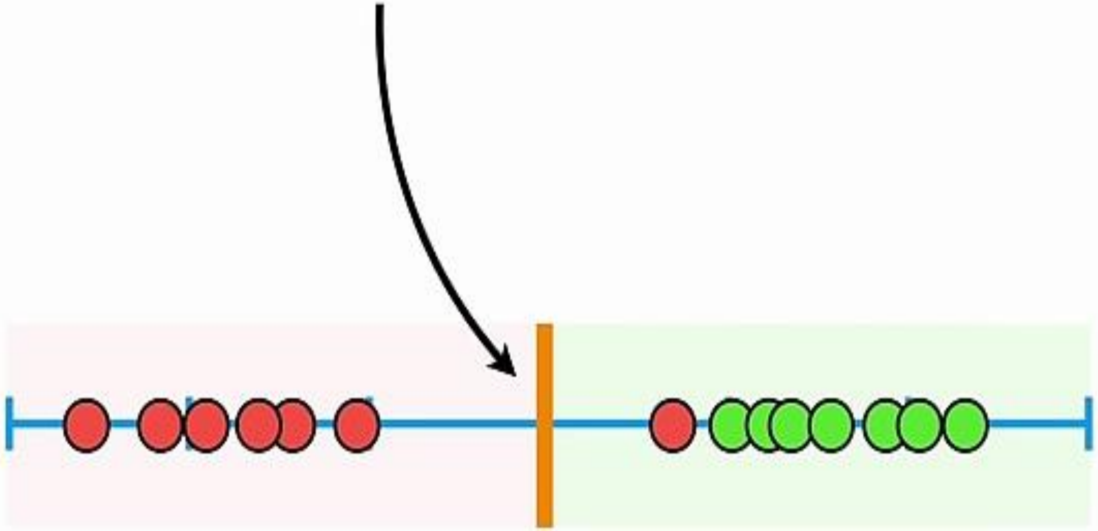


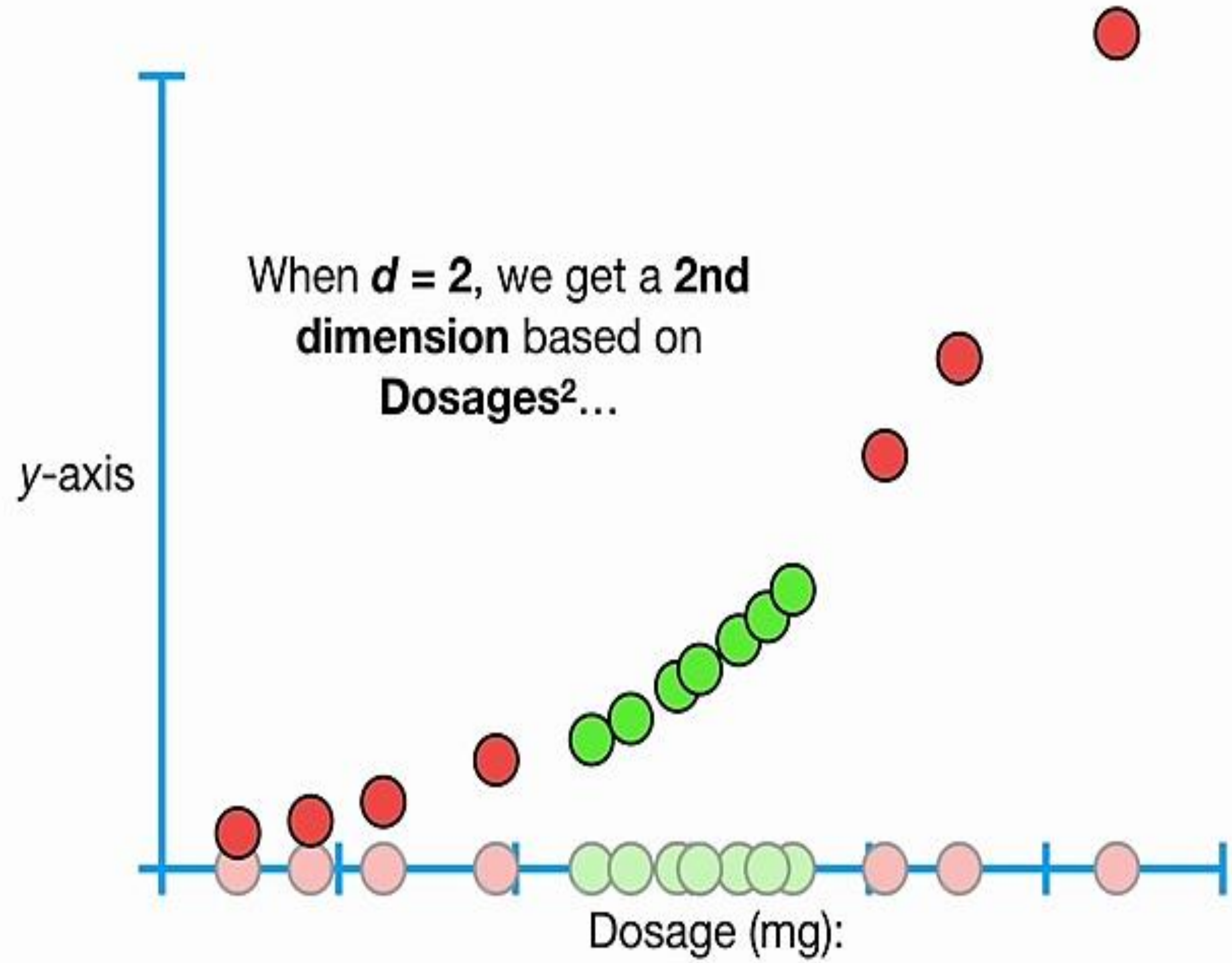
When ***d* = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…

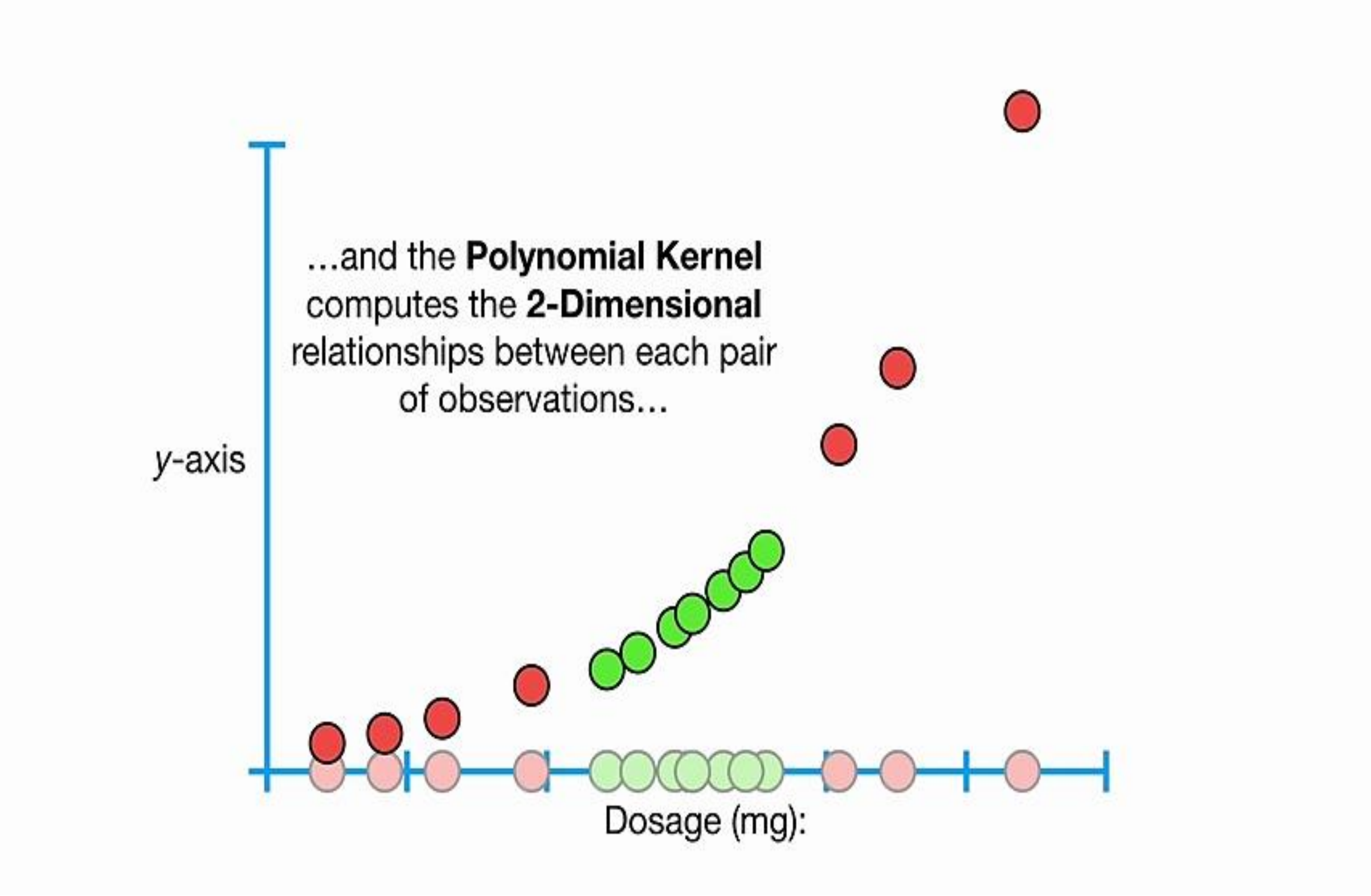


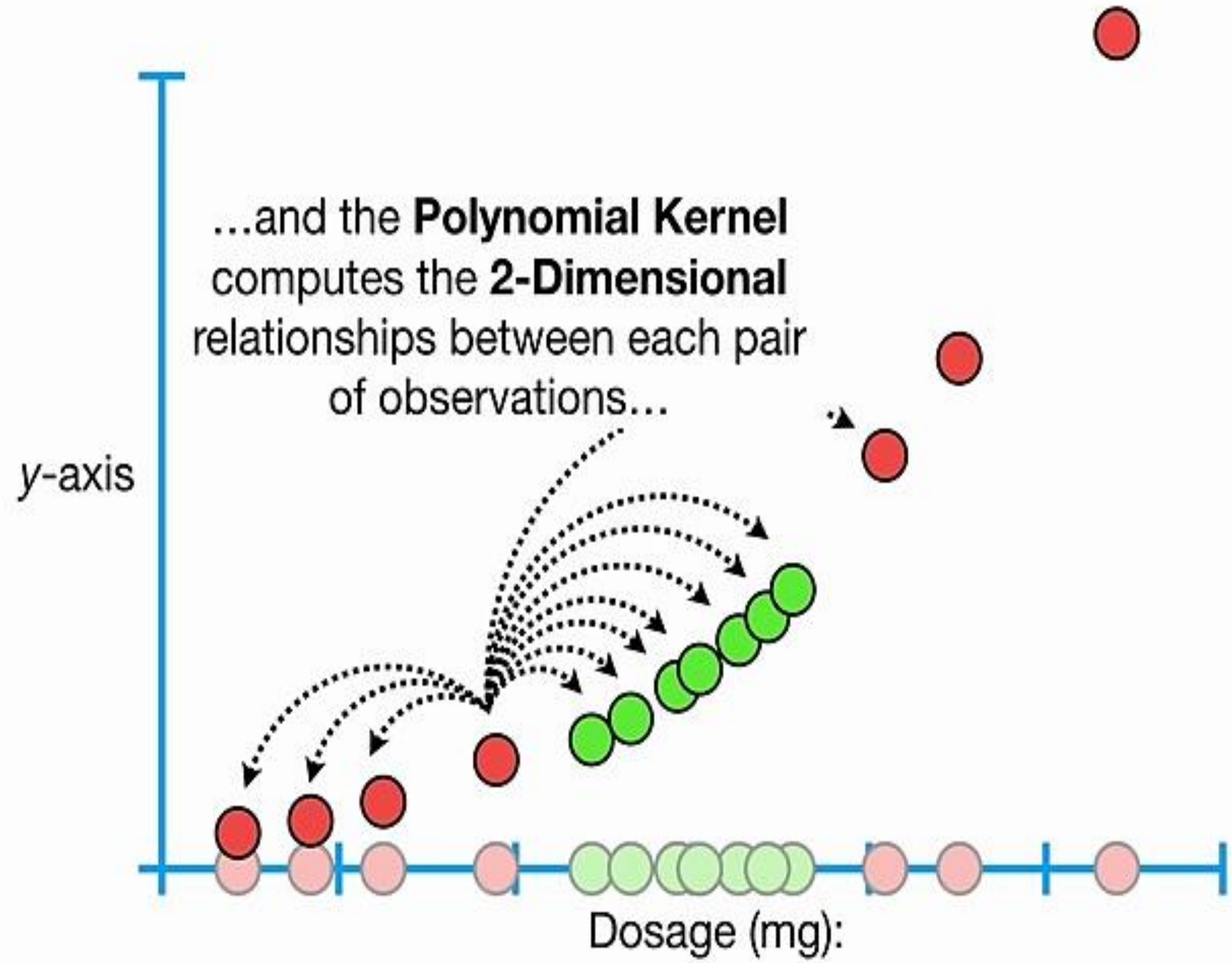
When ***d* = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…

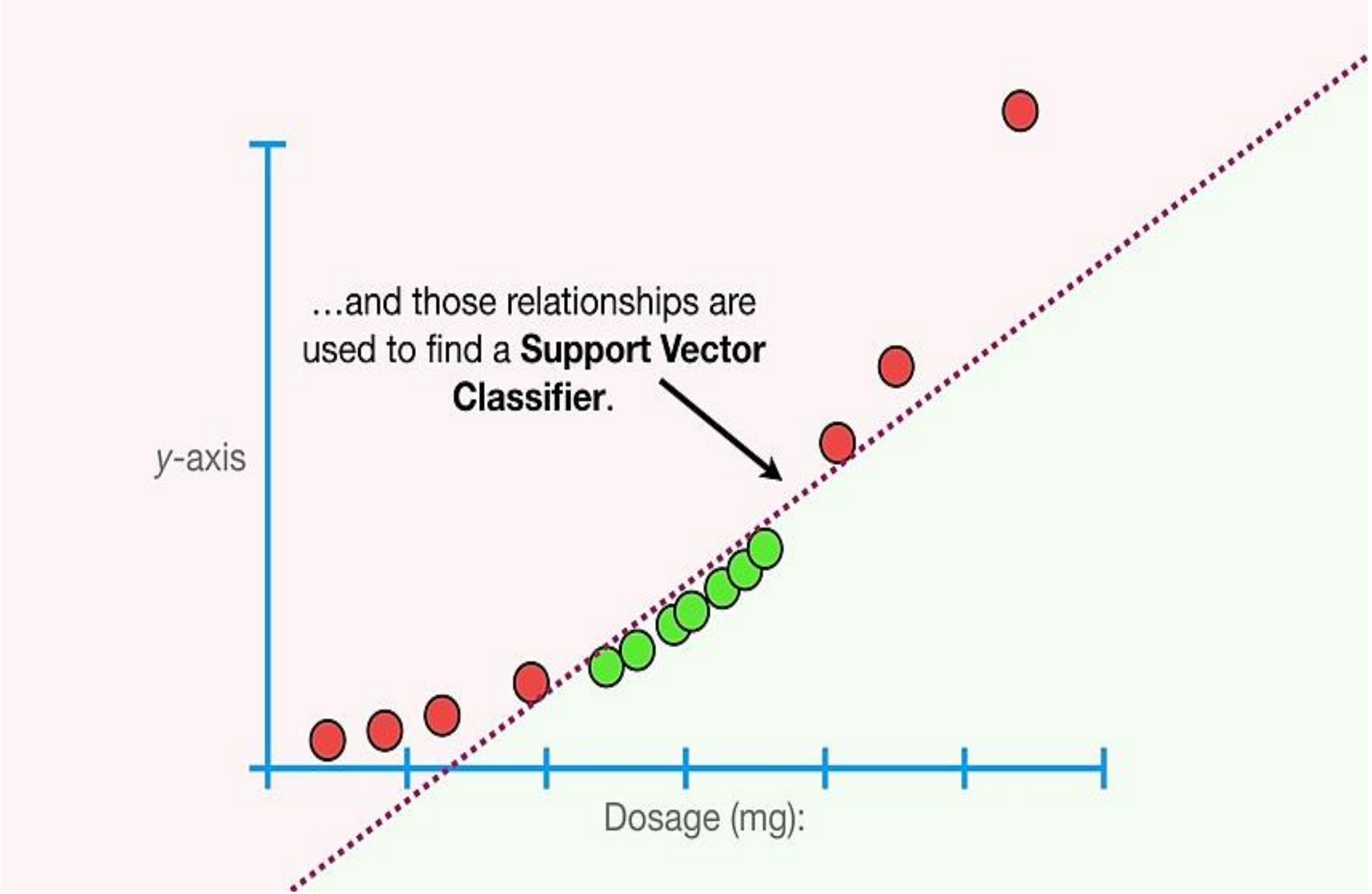
...and these relationships are used to find a **Support Vector Classifier**.

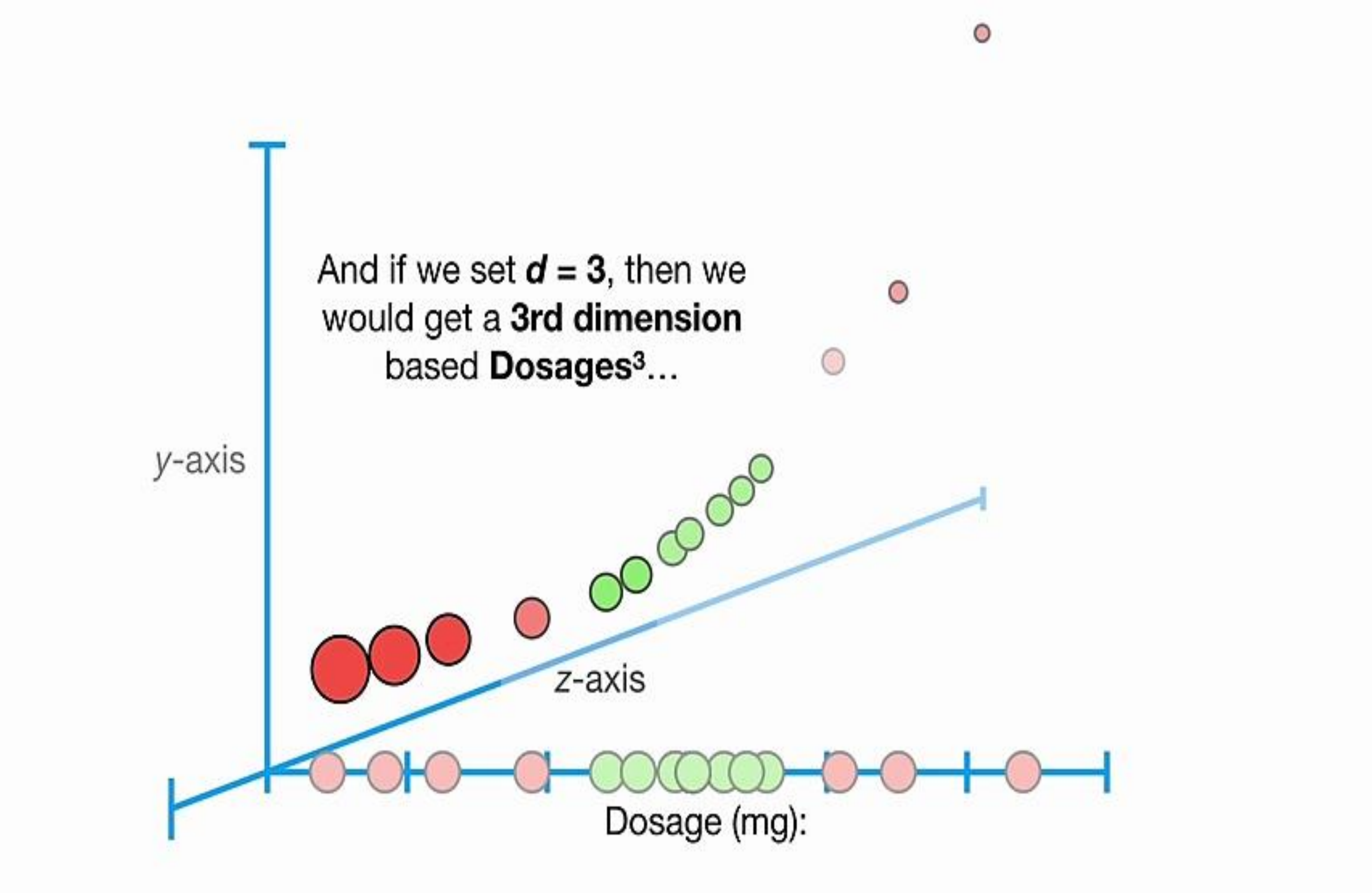


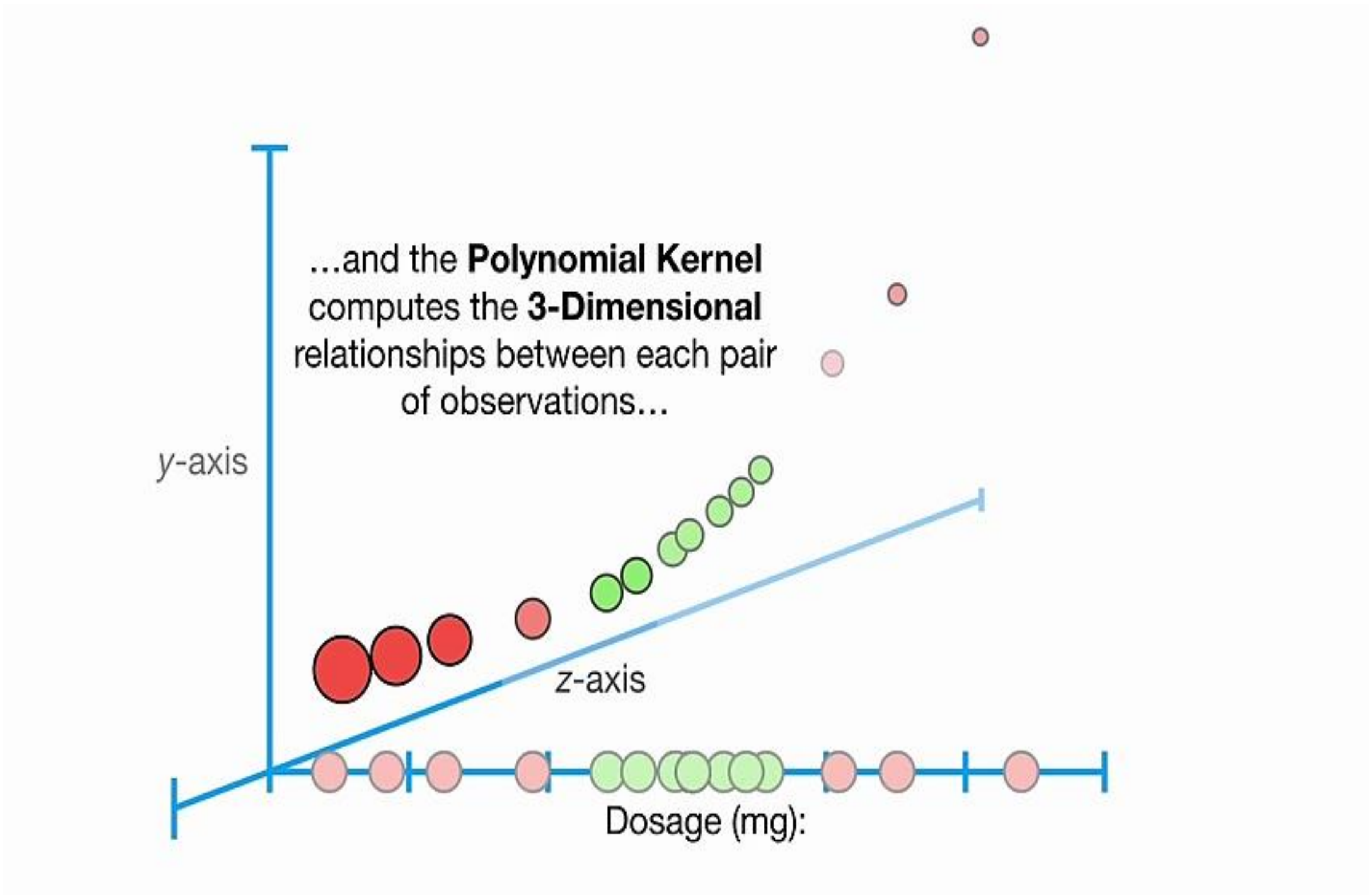

When d = 2, we get a 2nd dimension based on Dosages²...
y-axis
Dosage (mg):

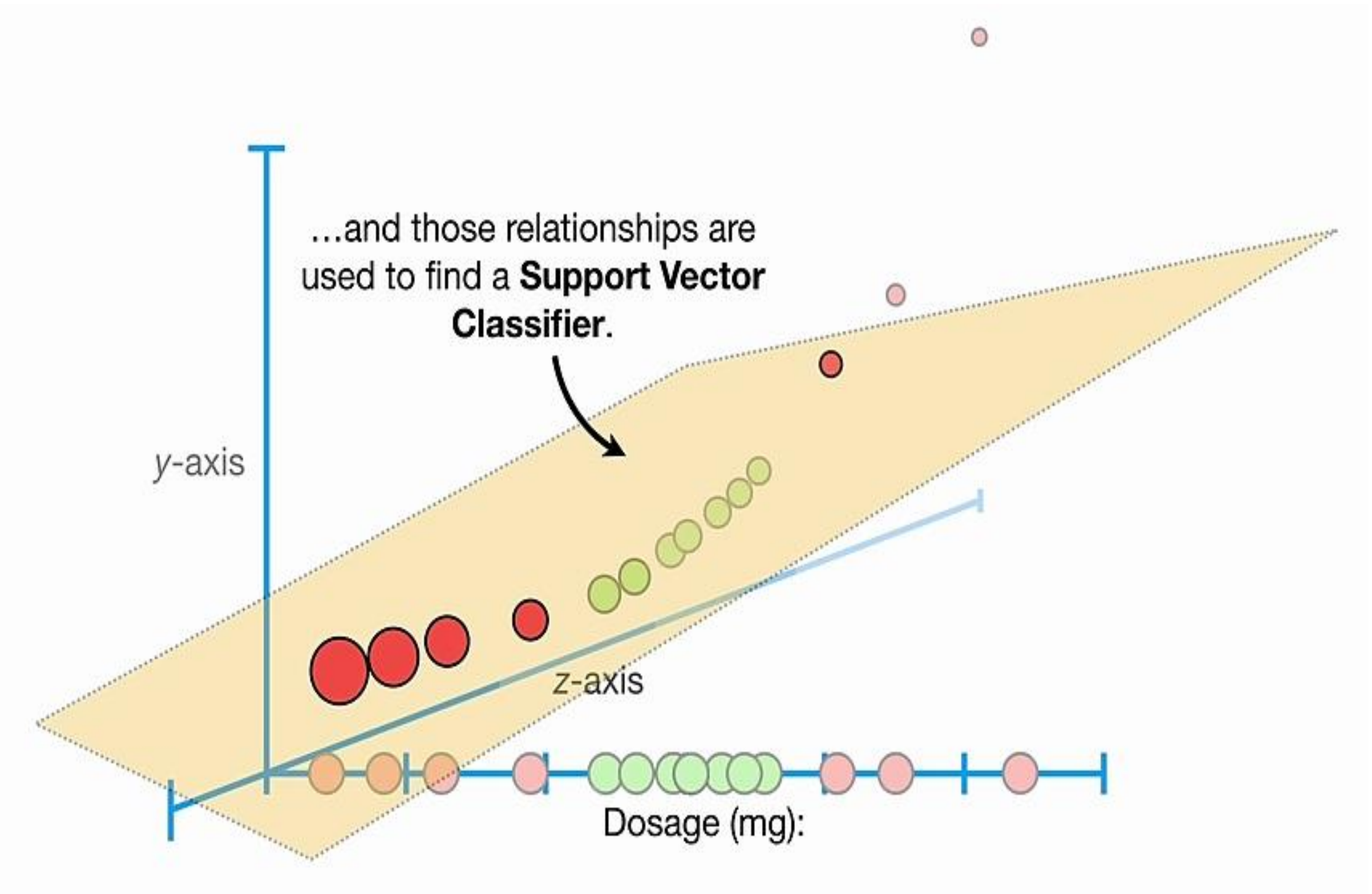

...and the **Polynomial Kernel** computes the **2-Dimensional** relationships between each pair of observations...
y-axis
Dosage (mg):


...and the **Polynomial Kernel** computes the **2-Dimensional** relationships between each pair of observations...
y-axis
Dosage (mg):


...and those relationships are used to find a **Support Vector Classifier.**
y-axis
Dosage (mg):


And if we set *d* = 3, then we would get a **3rd dimension** based **Dosages³**…
y-axis
z-axis
Dosage (mg):

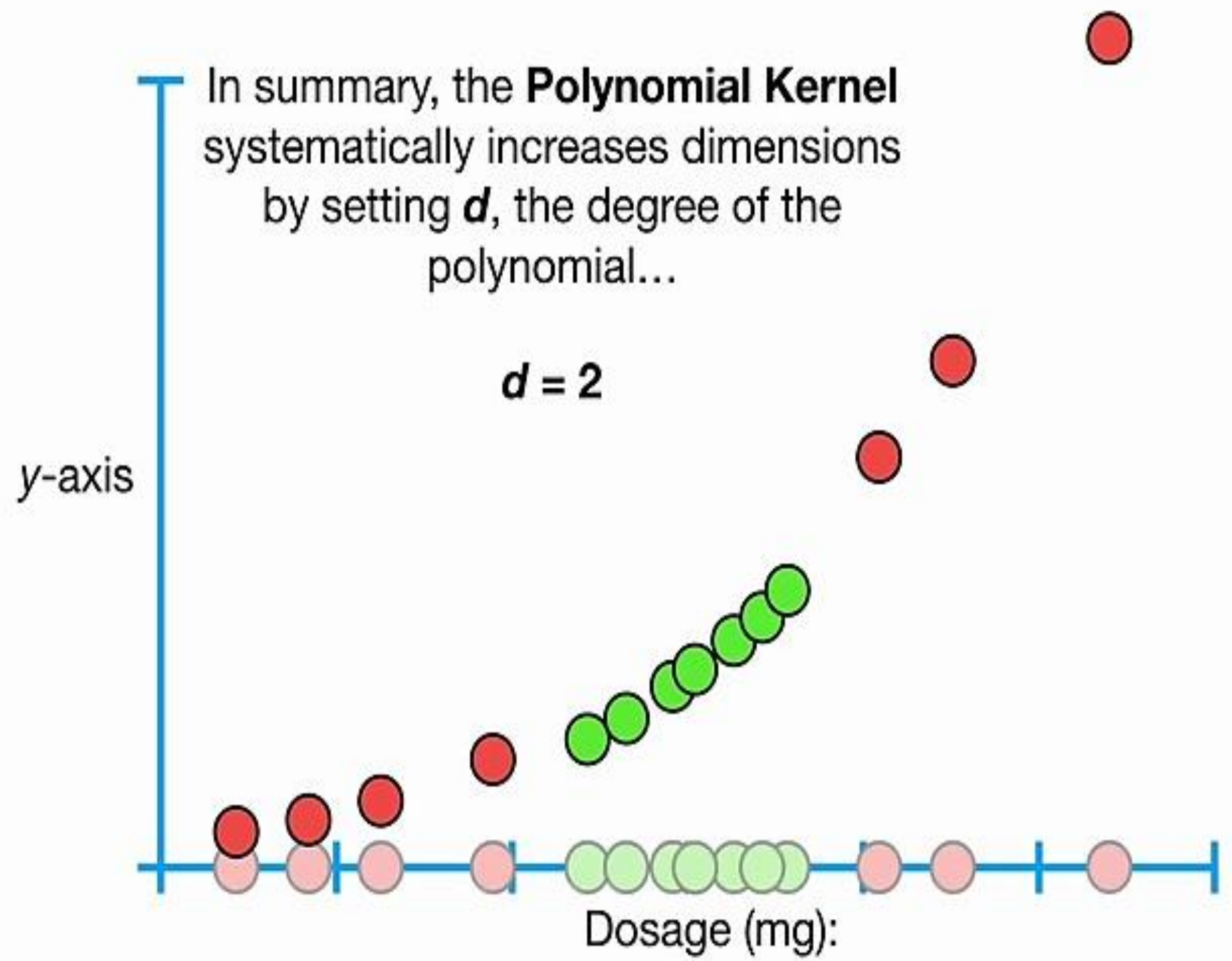

...and the **Polynomial Kernel** computes the **3-Dimensional** relationships between each pair of observations...
y-axis
z-axis
Dosage (mg):

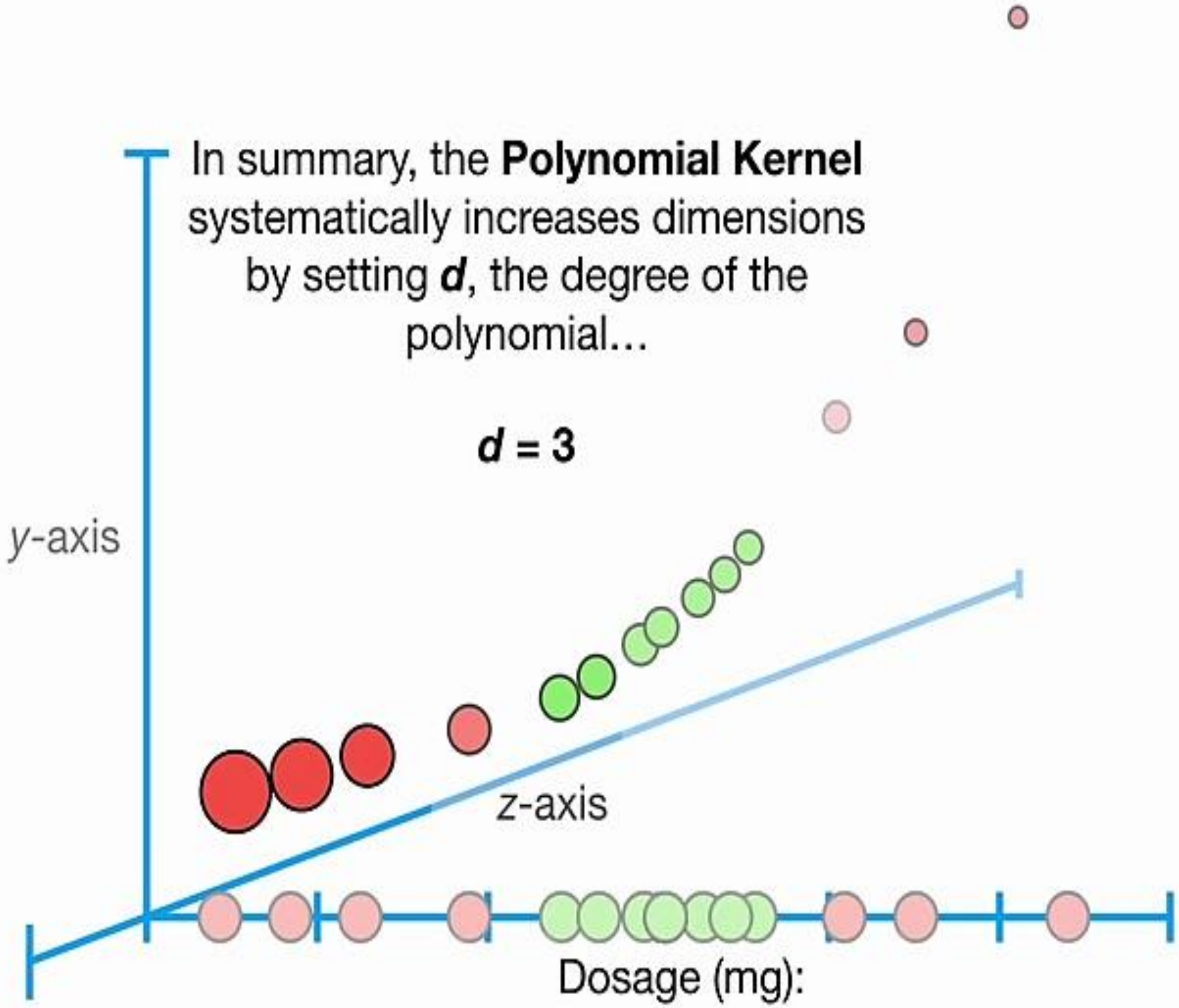

...and those relationships are used to find a **Support Vector Classifier**.
y-axis
z-axis
Dosage (mg):

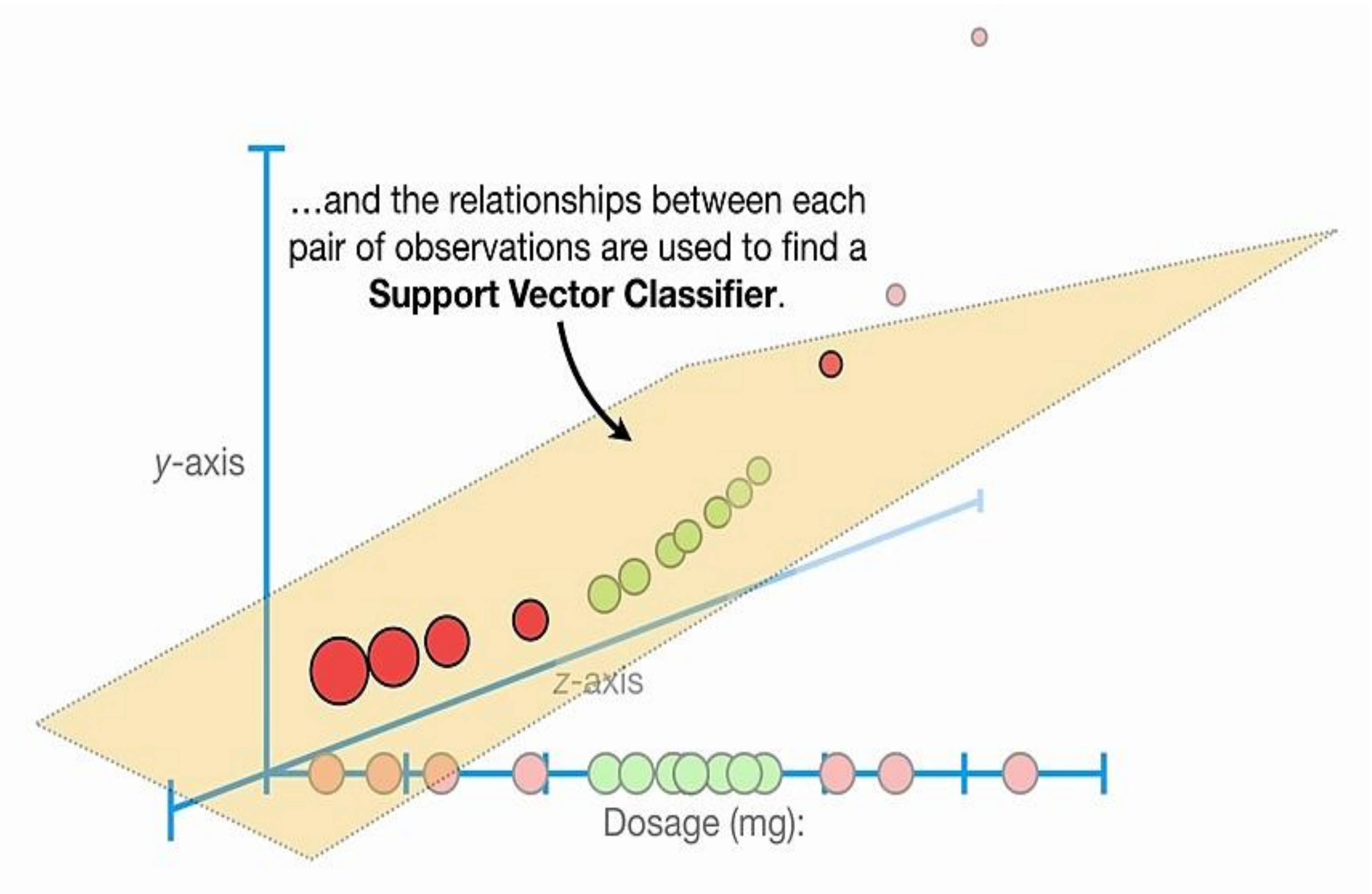
And when ***d* = 4 or more**, then we get even more dimensions to find a **Support Vector Classifier.**

In summary, the **Polynomial Kernel** systematically increases dimensions by setting ***d***, the degree of the polynomial...

***d* = 1**

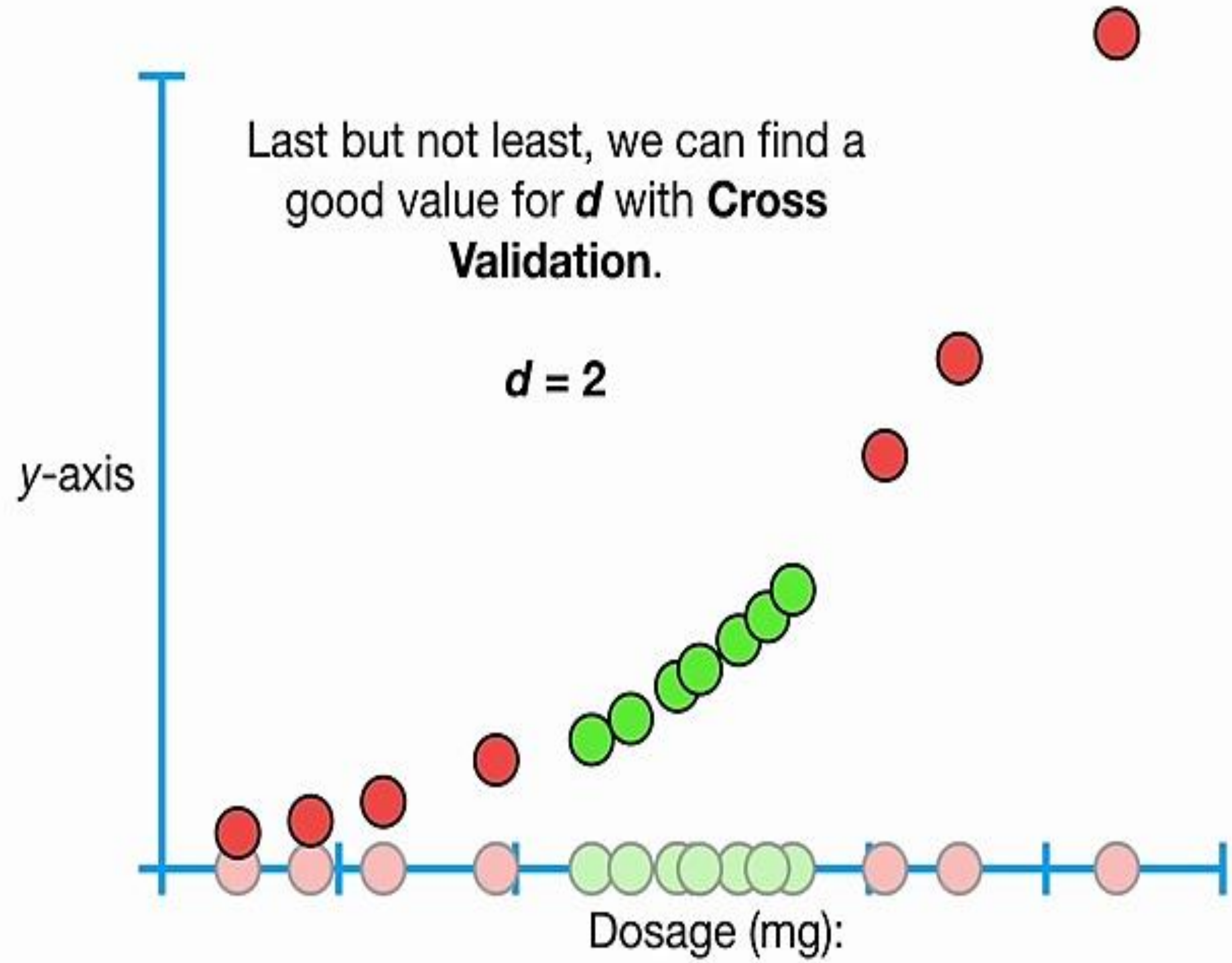

In summary, the **Polynomial Kernel** systematically increases dimensions by setting ***d***, the degree of the polynomial...
***d* = 2**
y-axis
Dosage (mg):

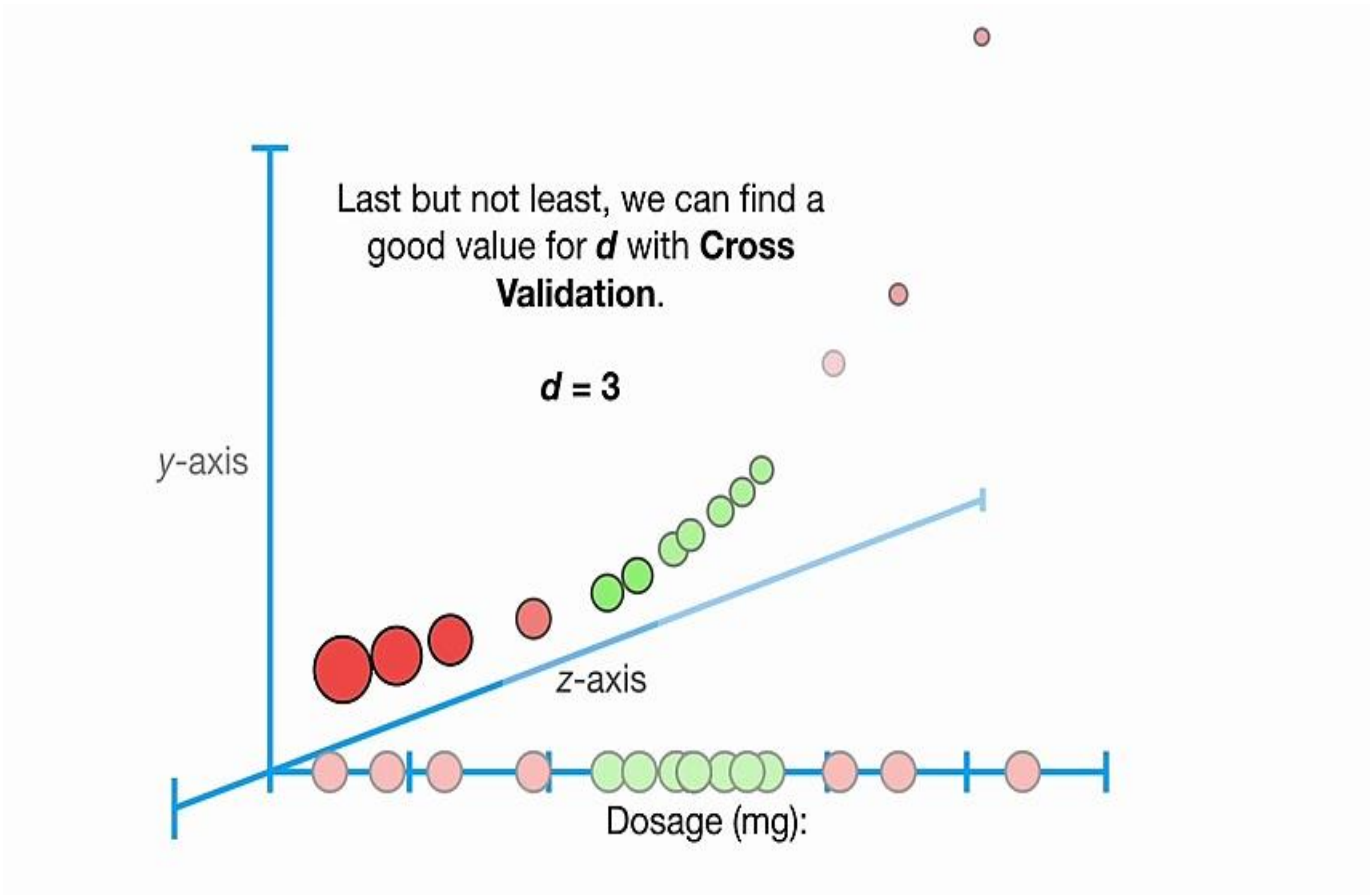

In summary, the **Polynomial Kernel** systematically increases dimensions by setting ***d***, the degree of the polynomial...
***d* = 3**
y-axis
z-axis
Dosage (mg):


...and the relationships between each pair of observations are used to find a **Support Vector Classifier**.
y-axis
z-axis
Dosage (mg):

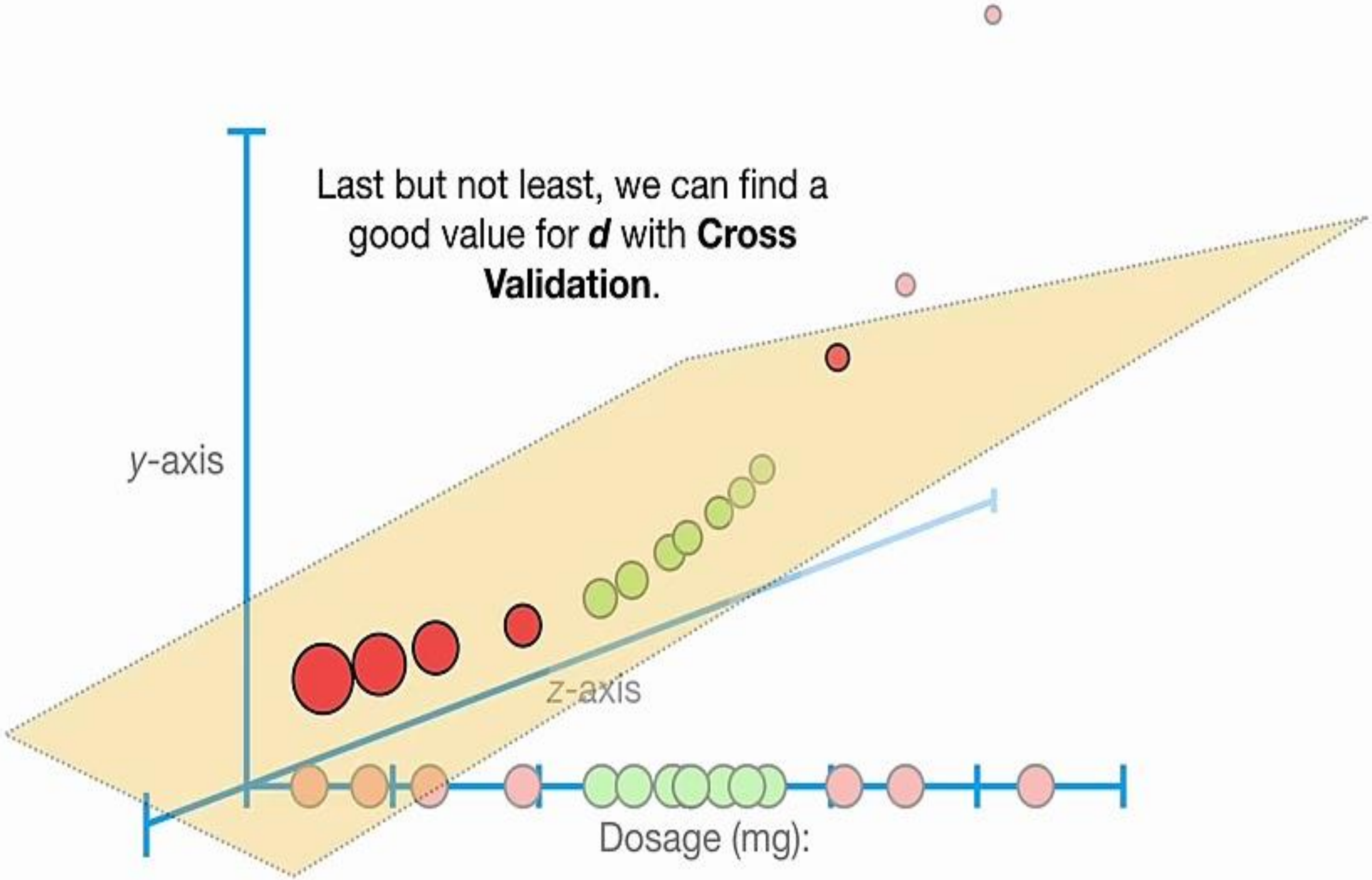
Last but not least, we can find a good value for ***d*** with **Cross Validation**.

***d* = 1**


Last but not least, we can find a good value for *d* with **Cross Validation**.
***d* = 2**
y-axis
Dosage (mg):

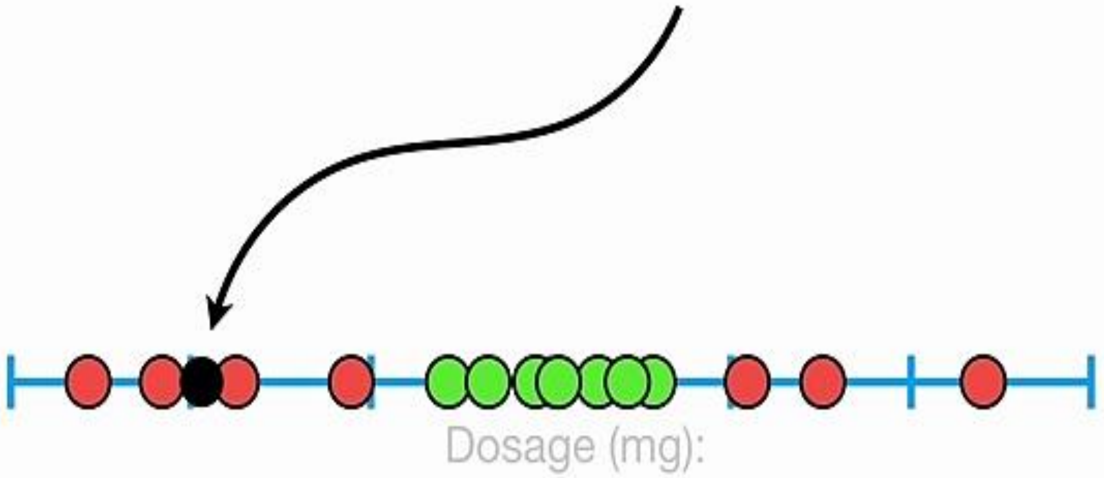


Last but not least, we can find a good value for ***d*** with **Cross Validation**.
***d* = 3**
y-axis
z-axis
Dosage (mg):


Last but not least, we can find a good value for ***d*** with **Cross Validation**.
y-axis
z-axis
Dosage (mg):

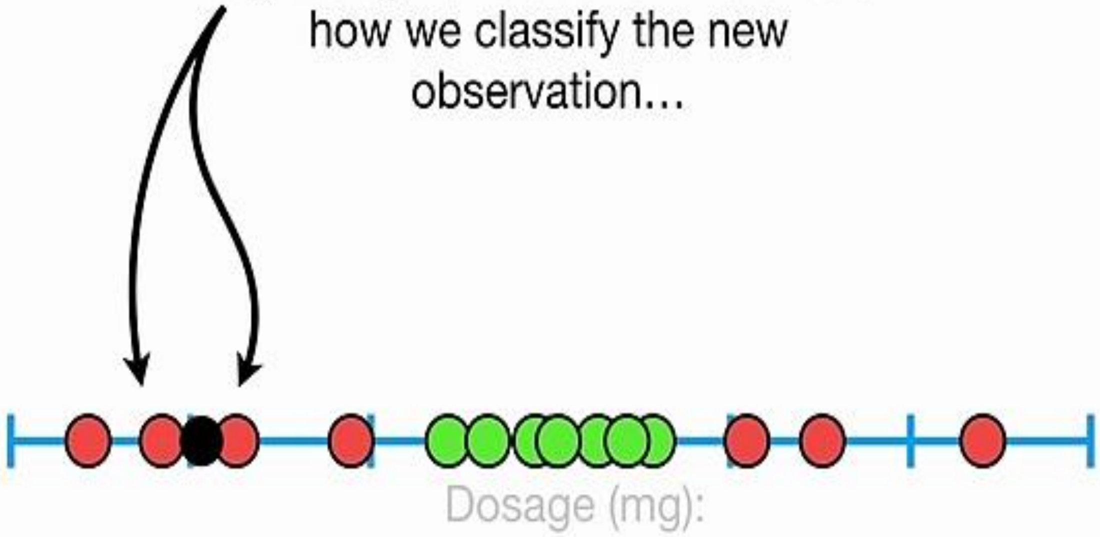
Another very commonly used **Kernel** is the **Radial Kernel**, also known as the **Radial Basis Function (RBF) Kernel**.

However, when using it on a new observation like this…

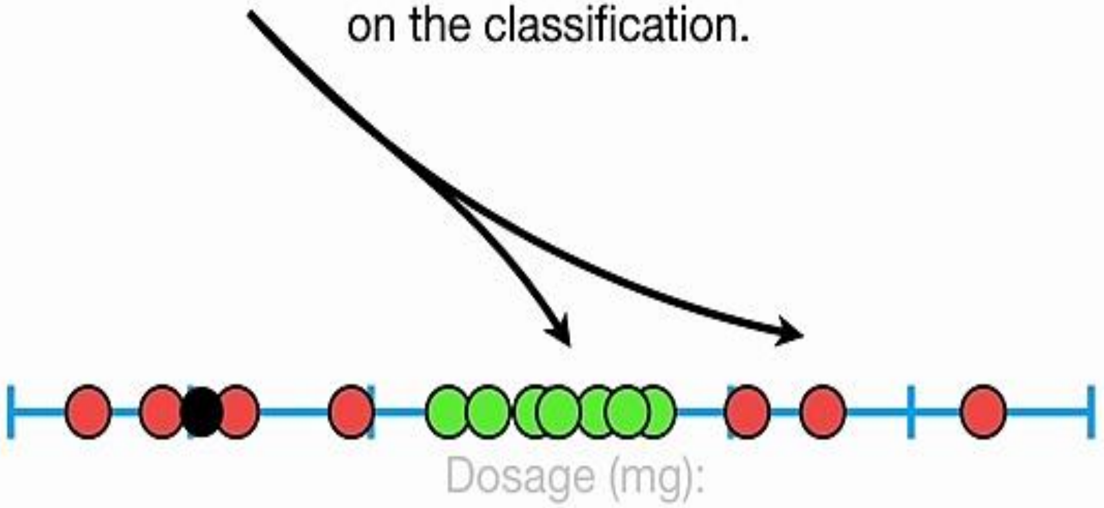


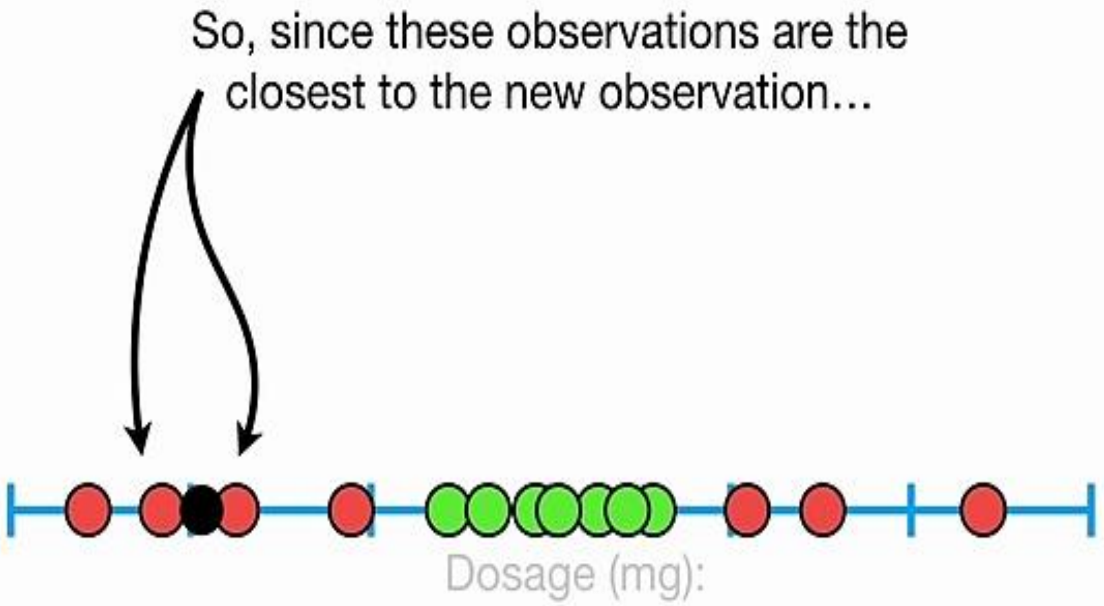
...the **Radial Kernel** behaves like a **Weighted Nearest Neighbor** model.

In other words, the closest observations (aka the nearest neighbors) have a lot of influence on how we classify the new observation…

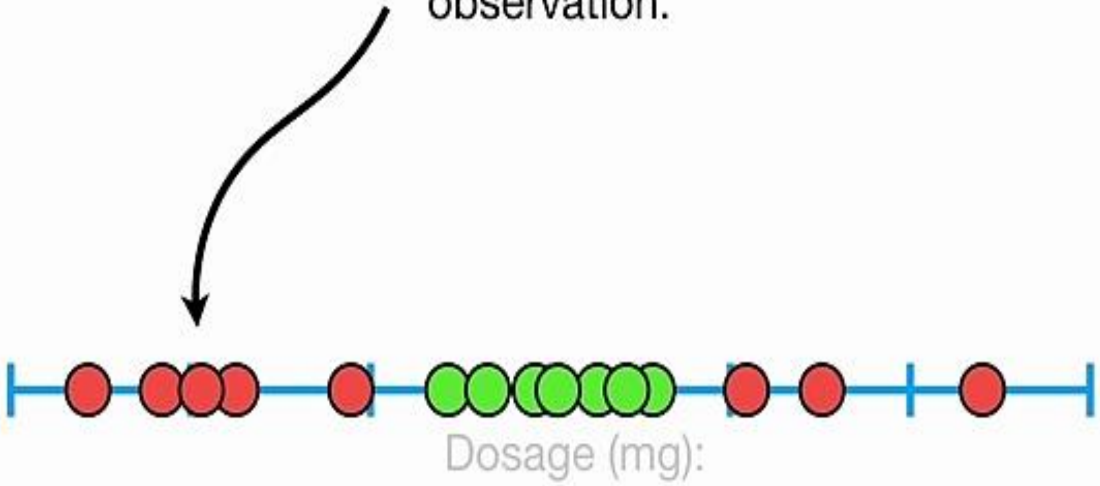


...and observations that are further away have relatively little influence on the classification.

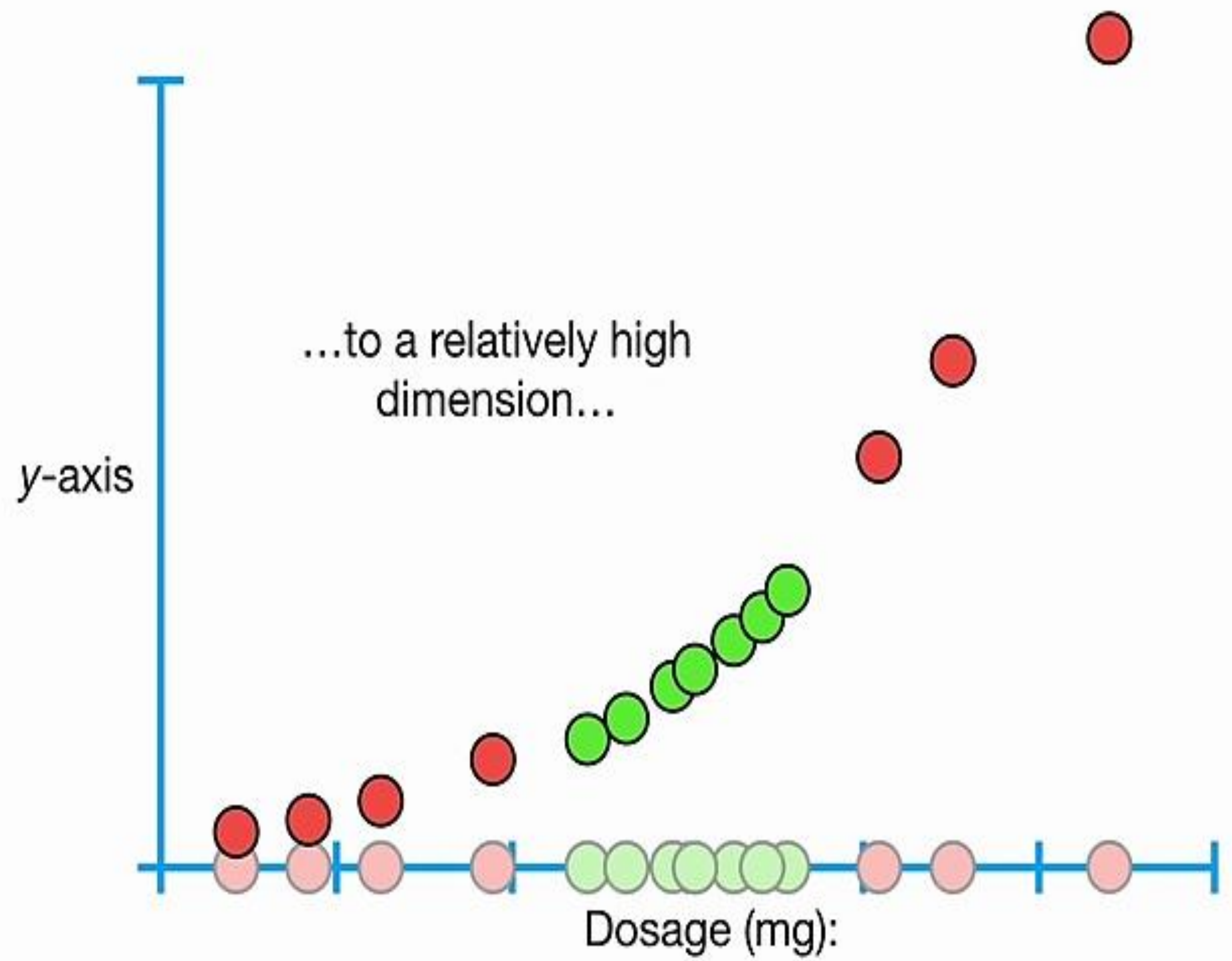


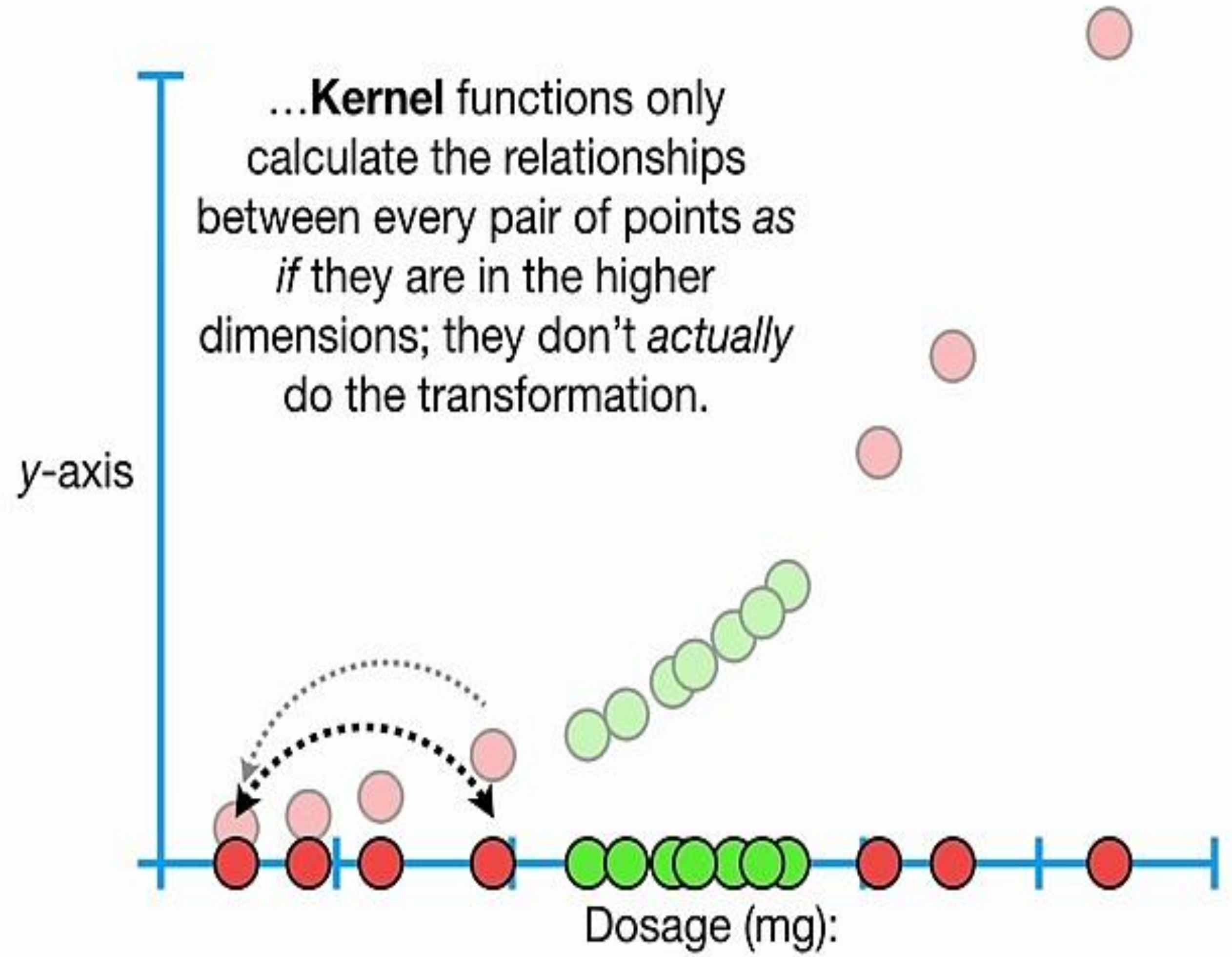

So, since these observations are the closest to the new observation...
Dosage (mg):

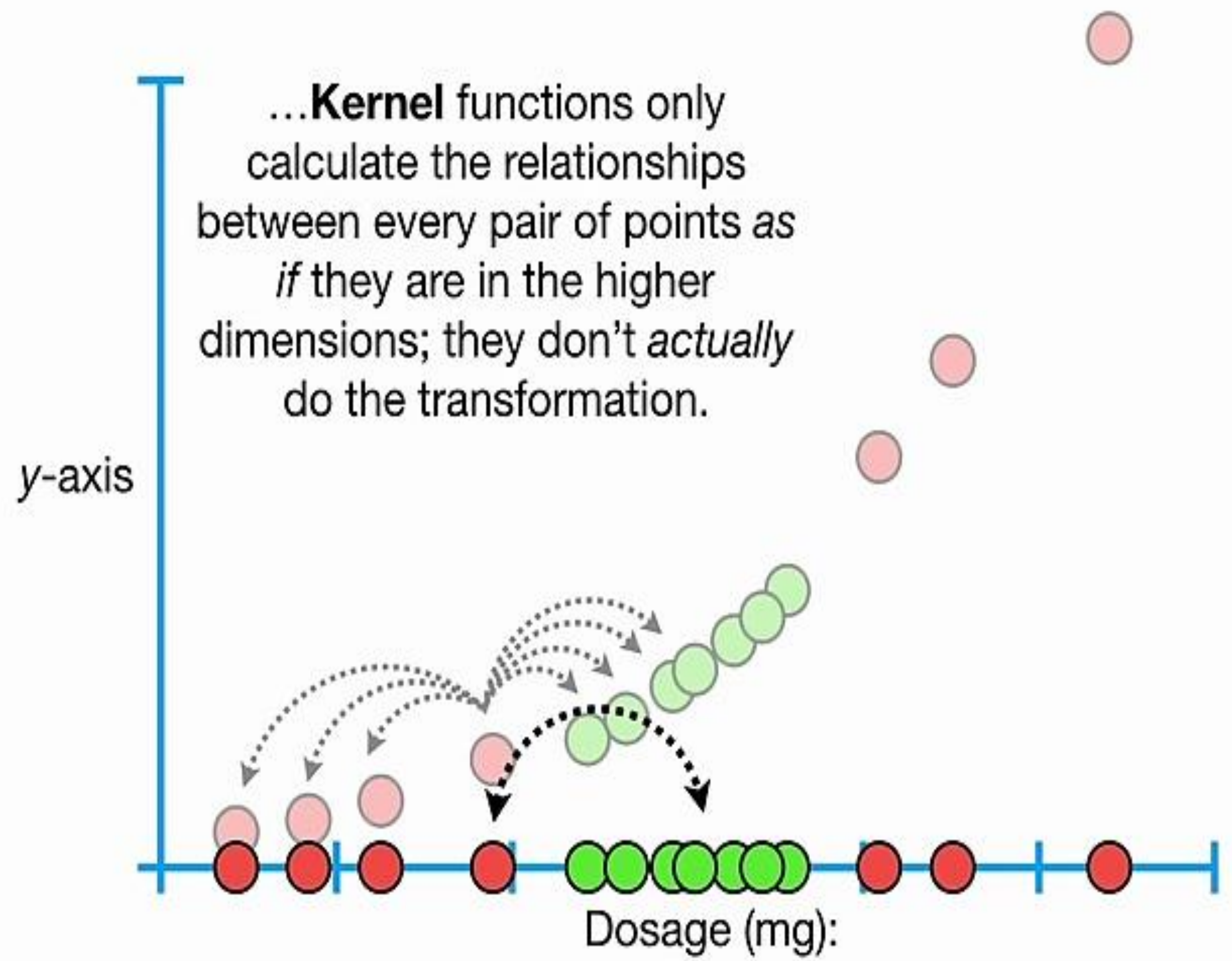
...the **Radial Kernel** uses their classification for the new observation.

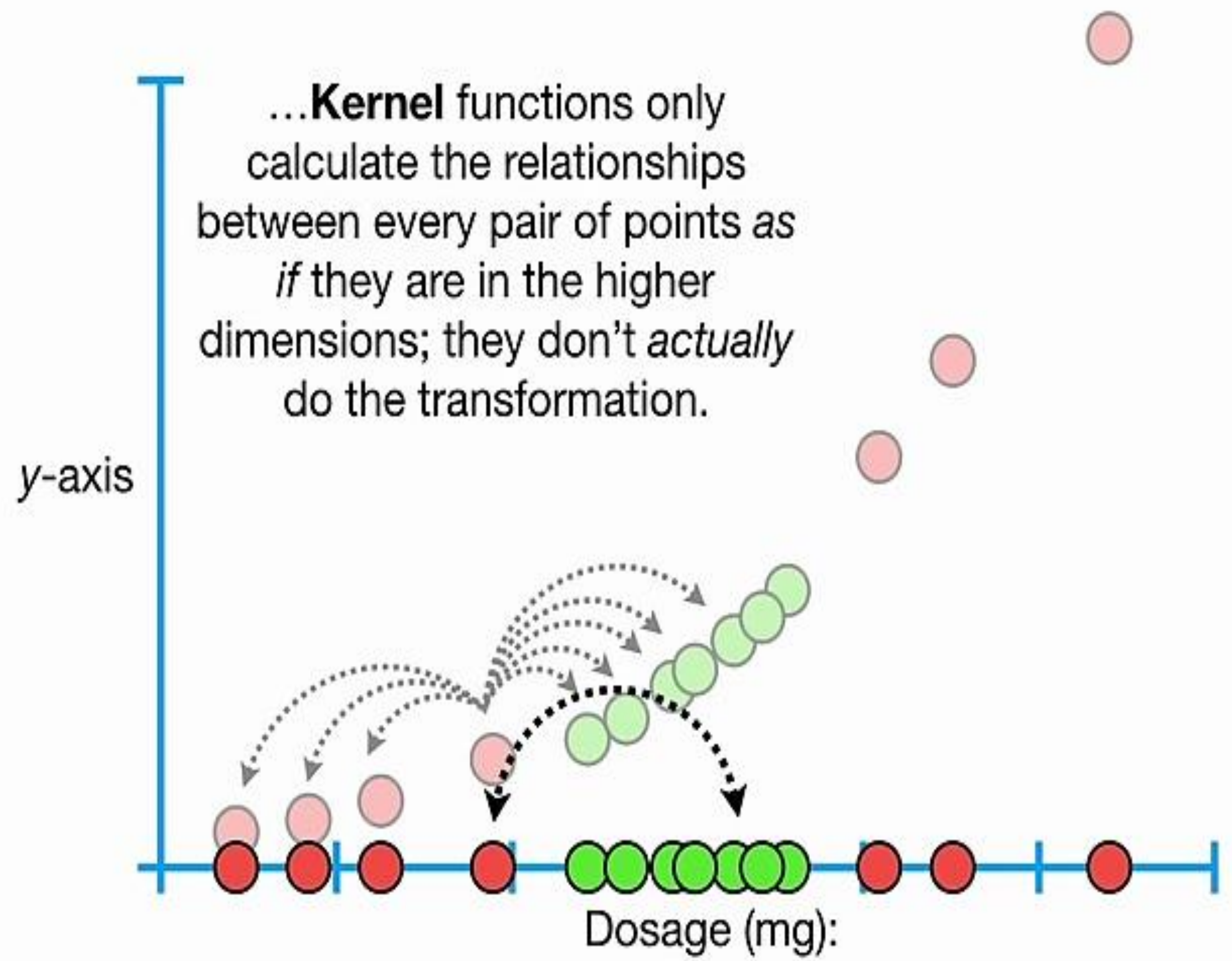


Although the examples I have given show the data being transformed from a relatively low dimension...

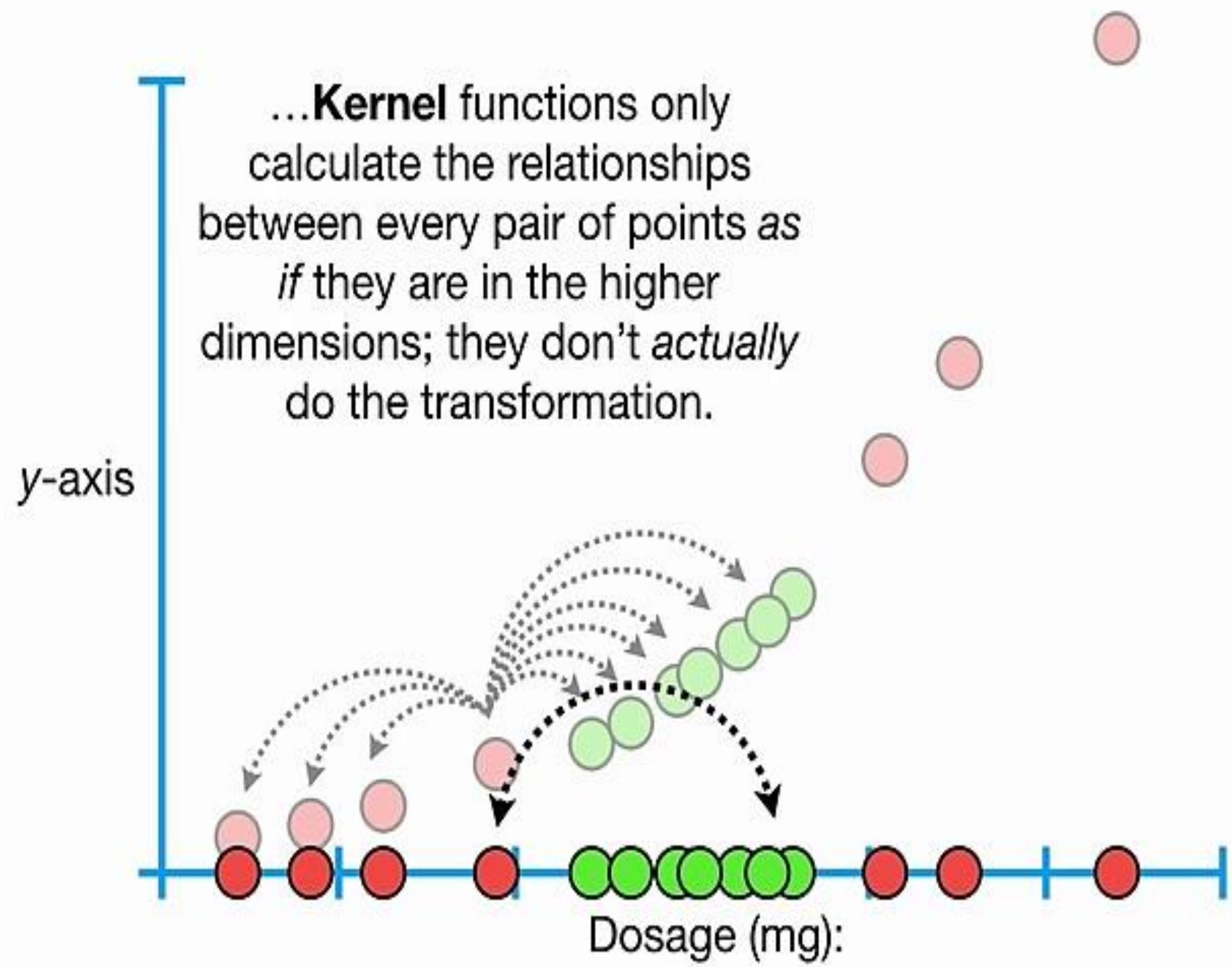

...to a relatively high dimension...
y-axis
Dosage (mg):

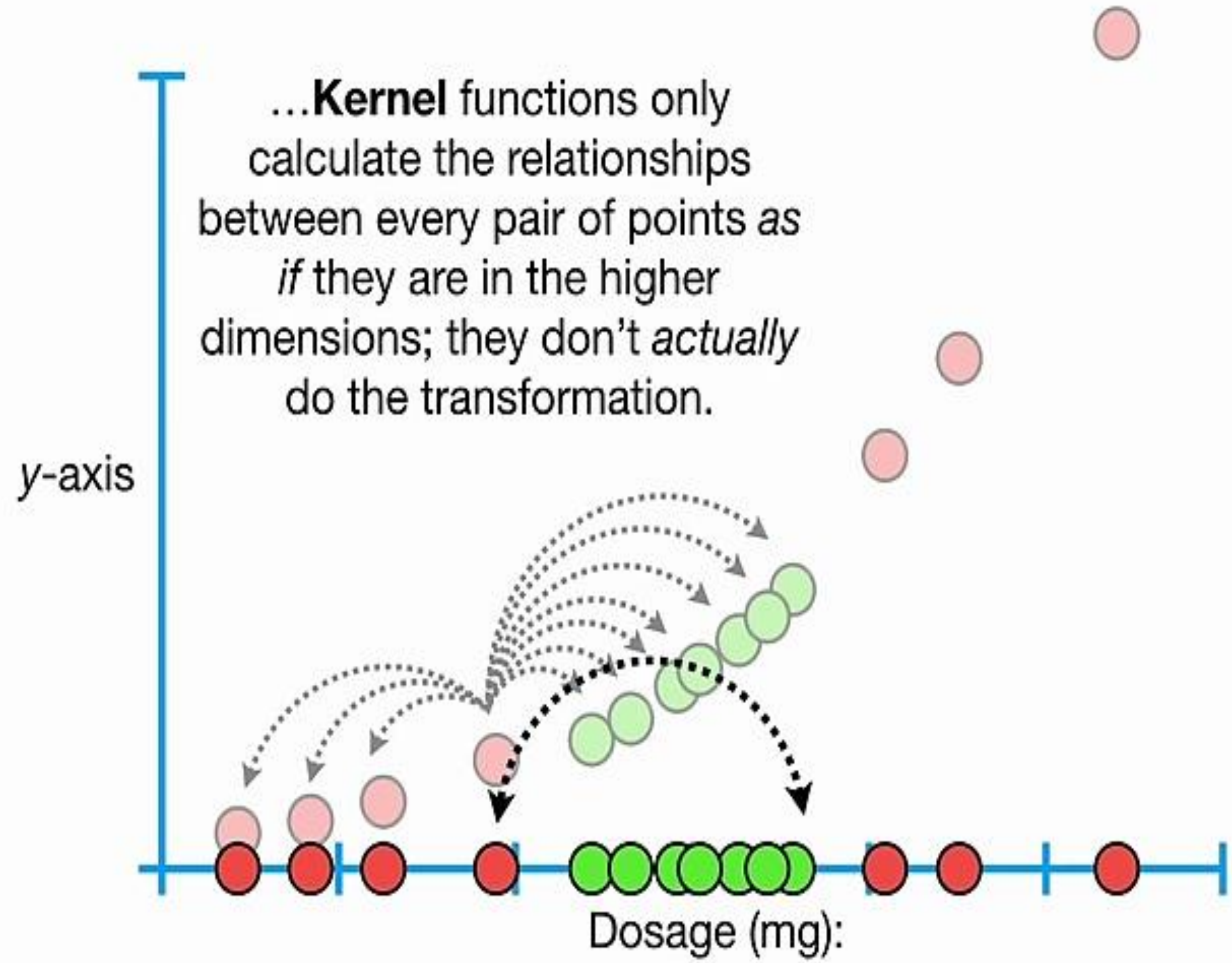

…**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):

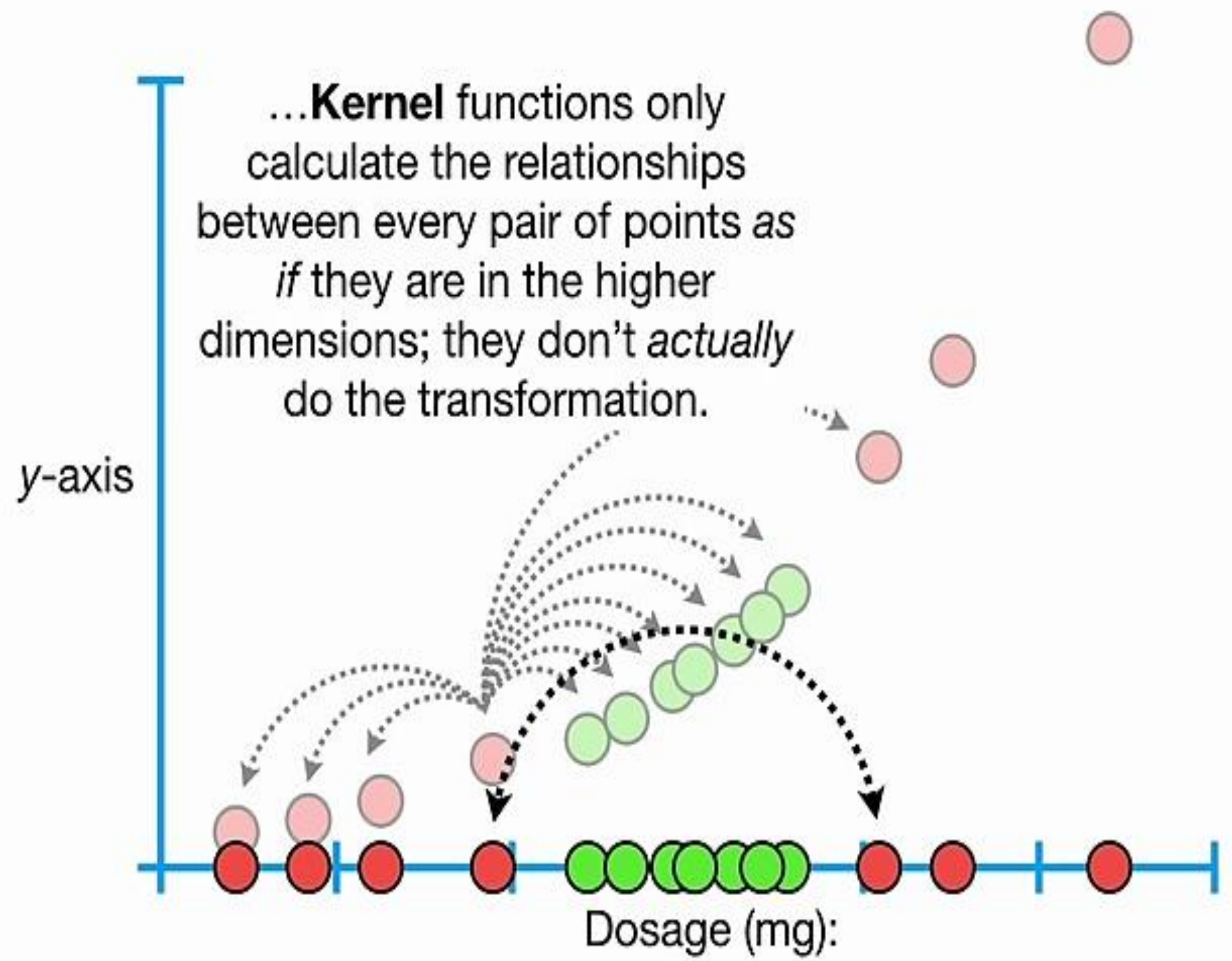

...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):

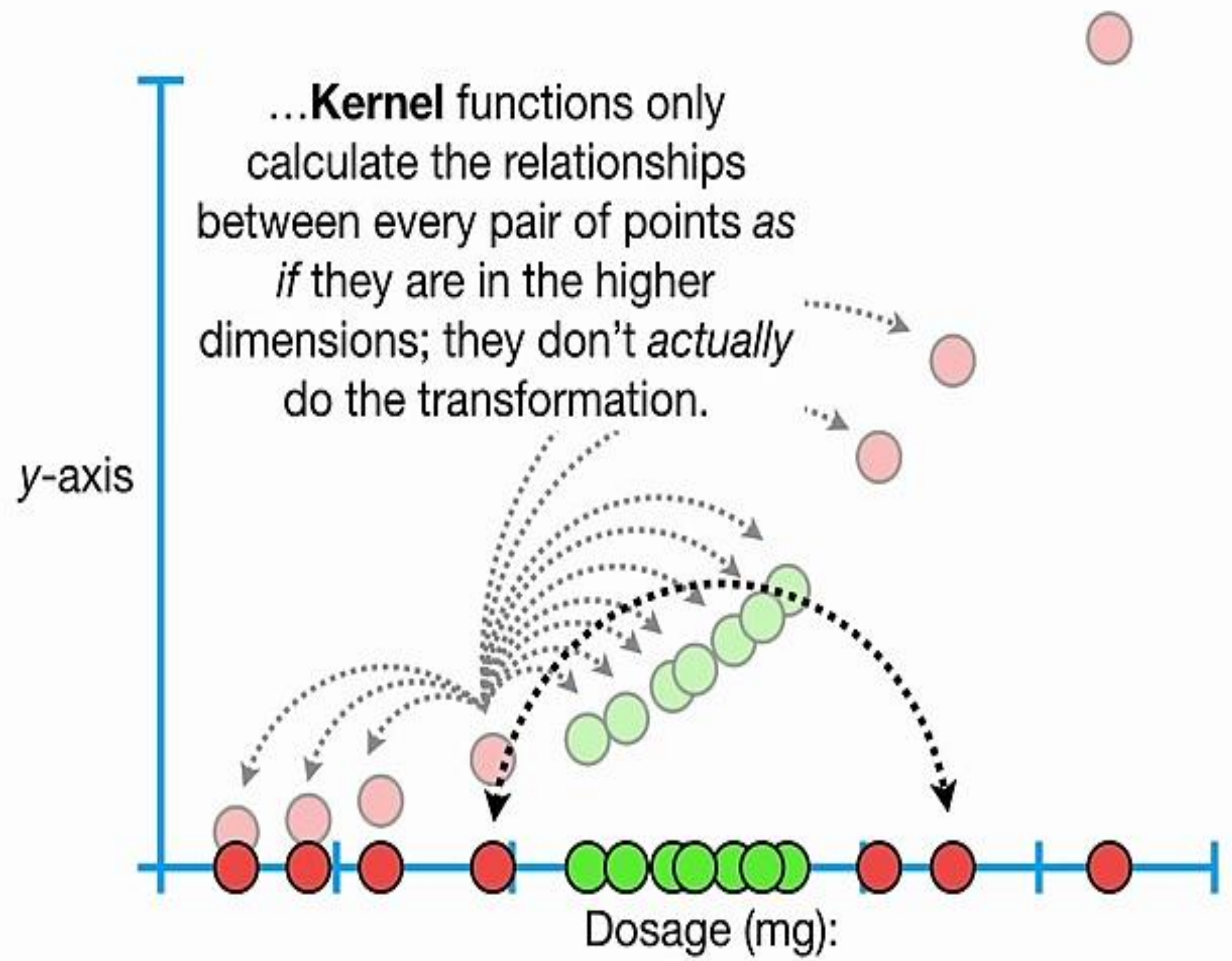


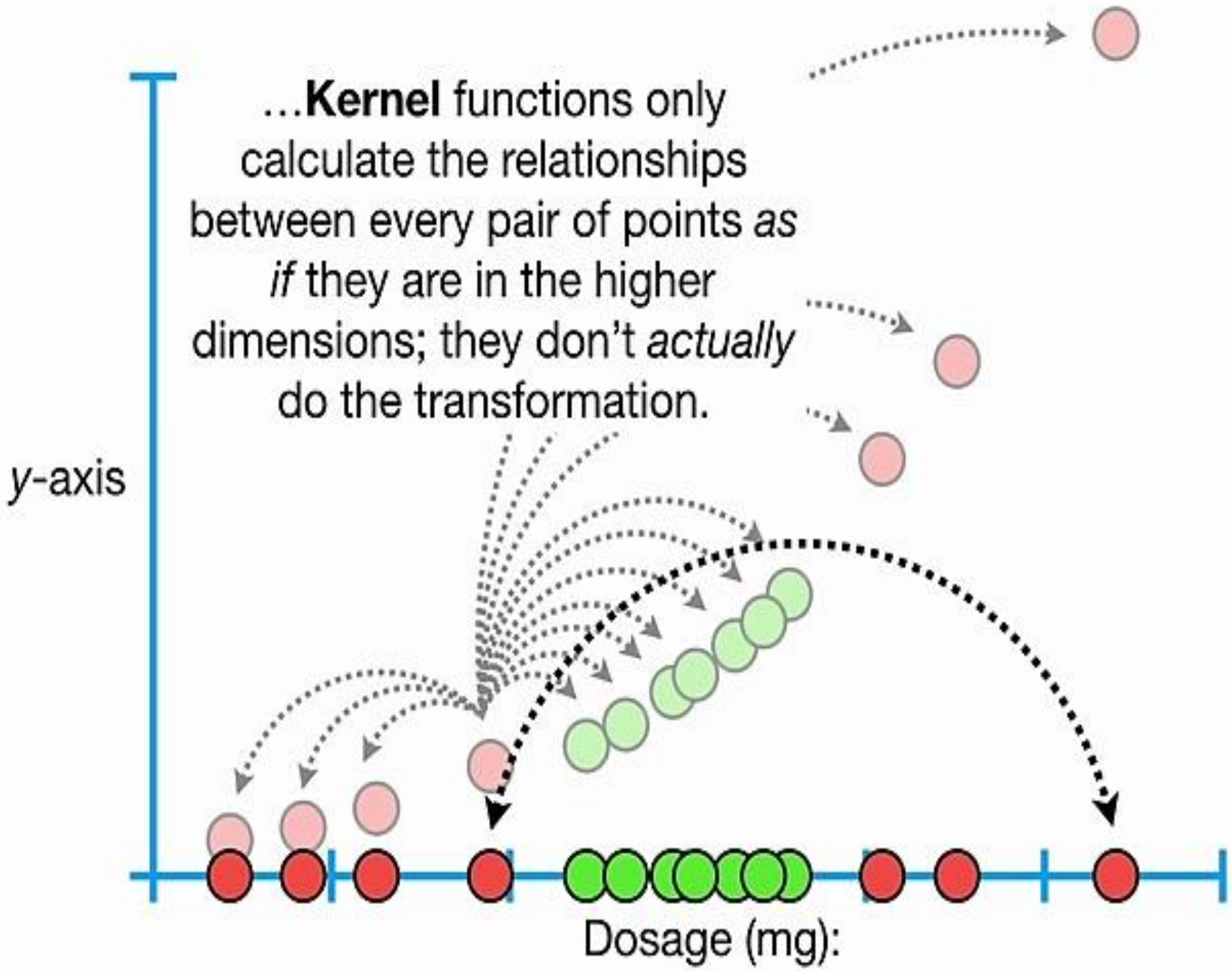
...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):


...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):

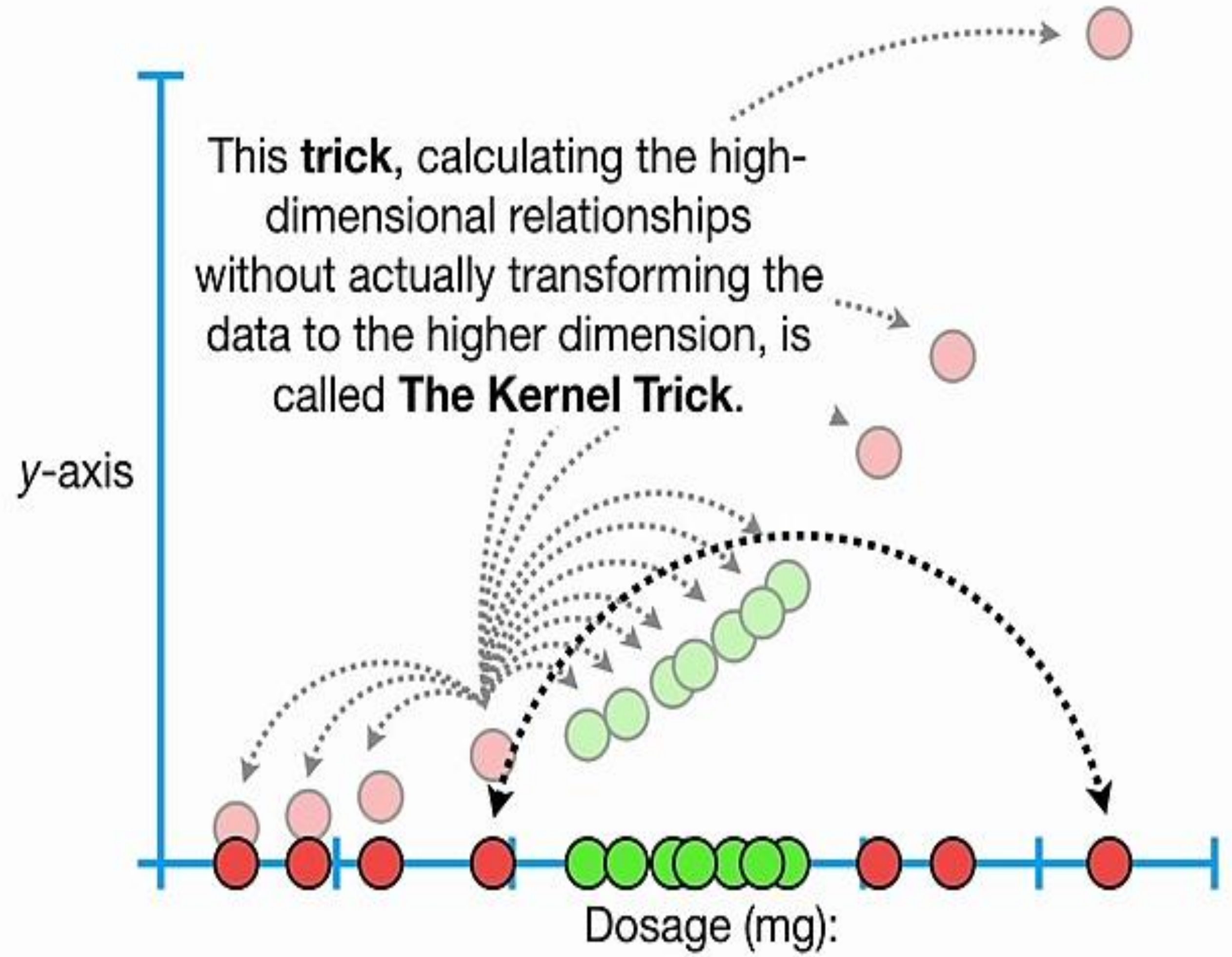

...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):

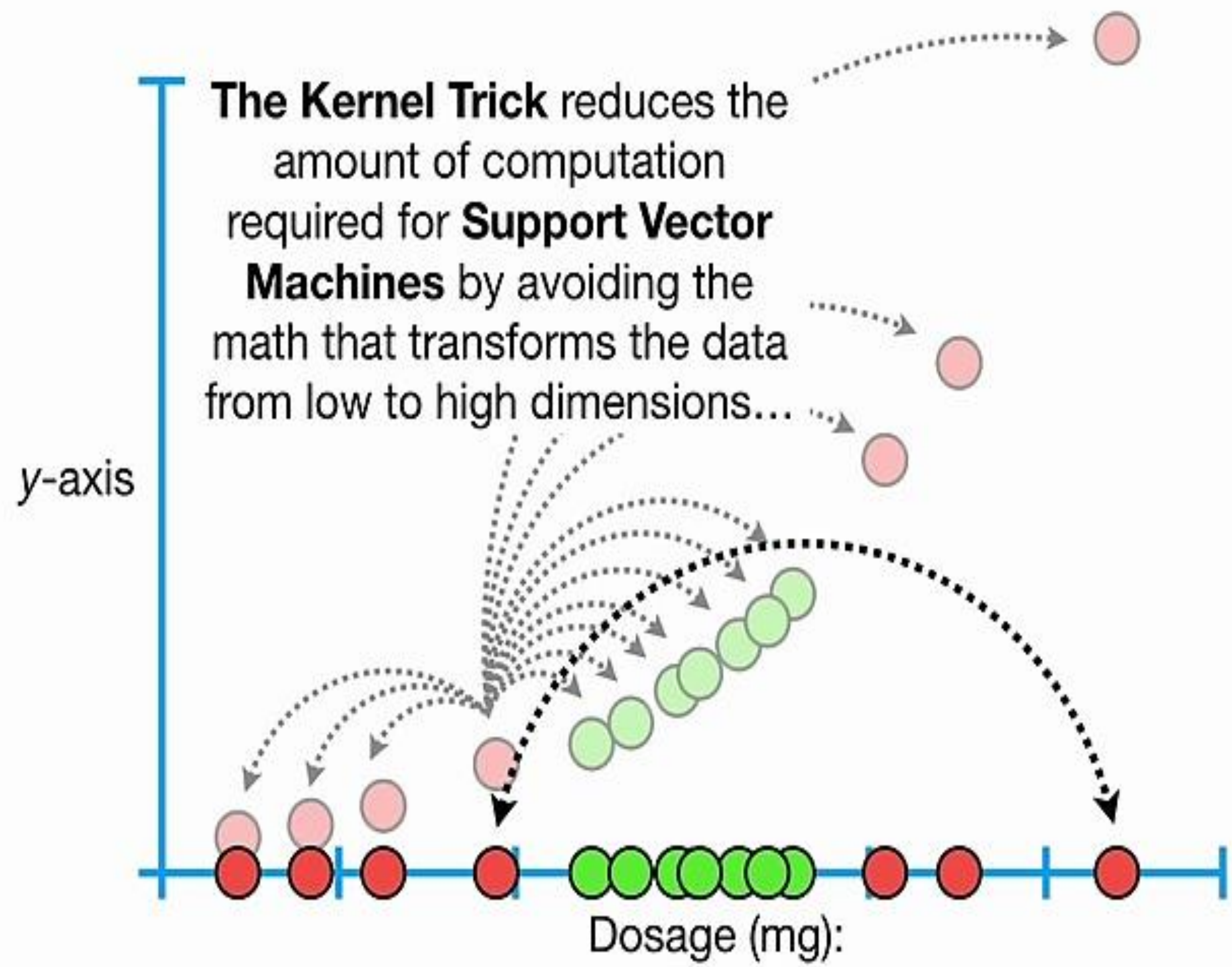

...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):


...**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):

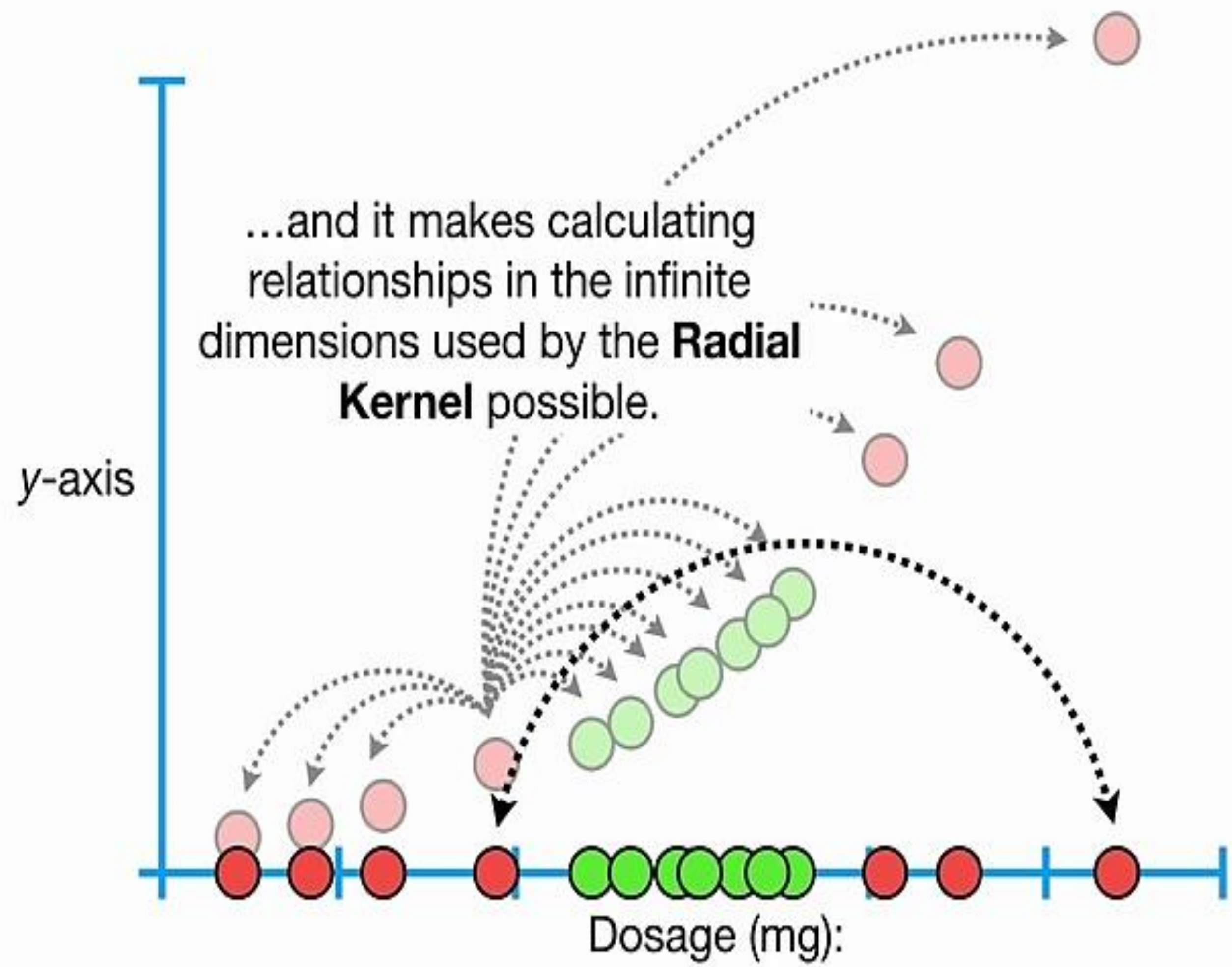


…**Kernel** functions only calculate the relationships between every pair of points *as if* they are in the higher dimensions; they don't *actually* do the transformation.
y-axis
Dosage (mg):


This **trick**, calculating the high-dimensional relationships without actually transforming the data to the higher dimension, is called **The Kernel Trick.**
y-axis
Dosage (mg):



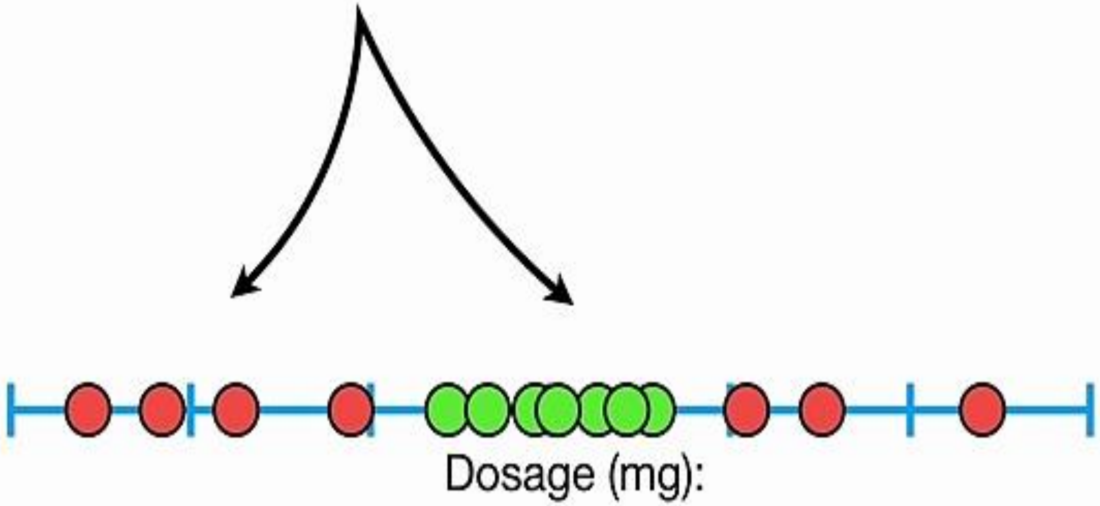
The Kernel Trick reduces the amount of computation required for **Support Vector Machines** by avoiding the math that transforms the data from low to high dimensions…
y-axis
Dosage (mg):

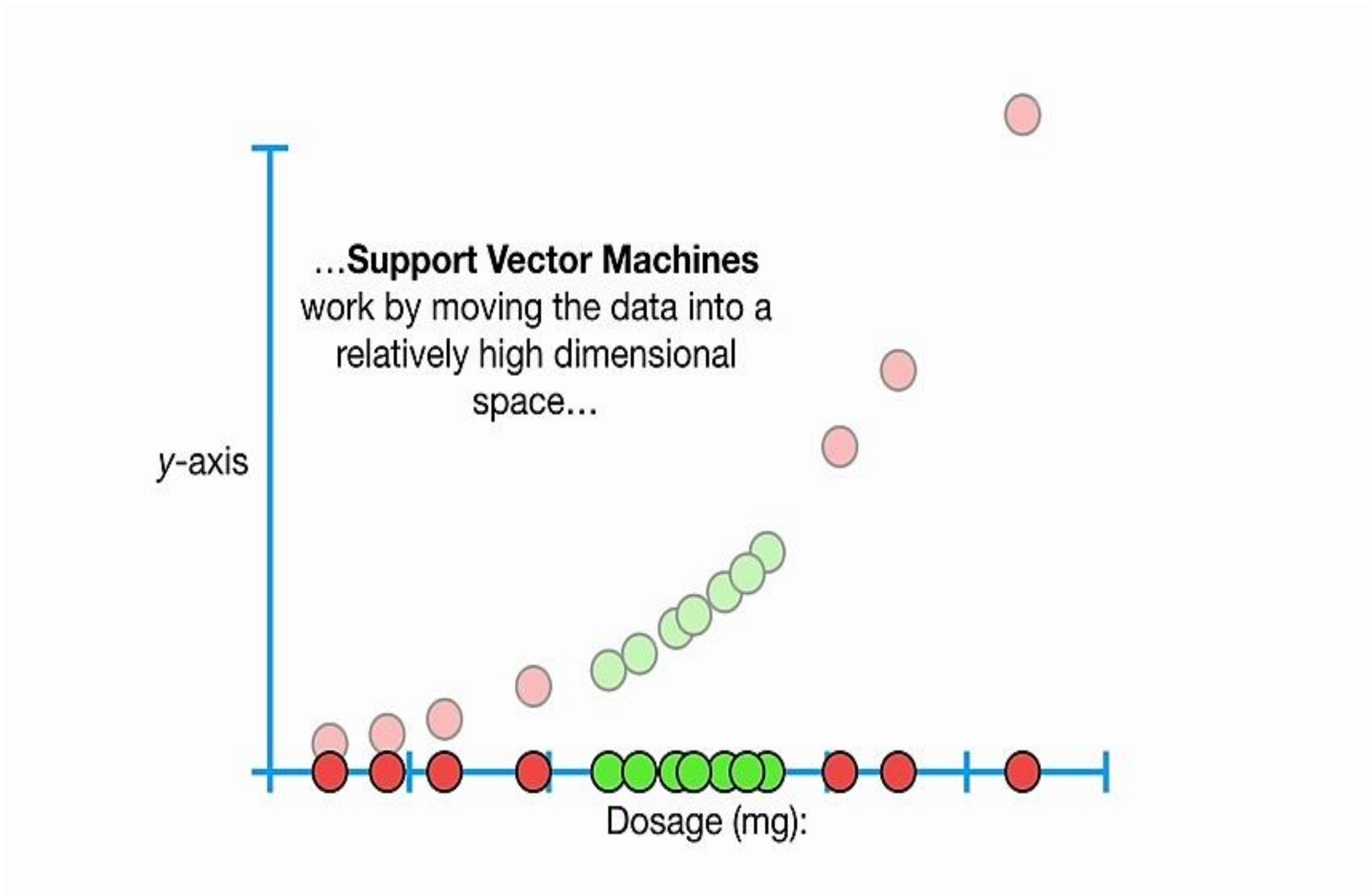


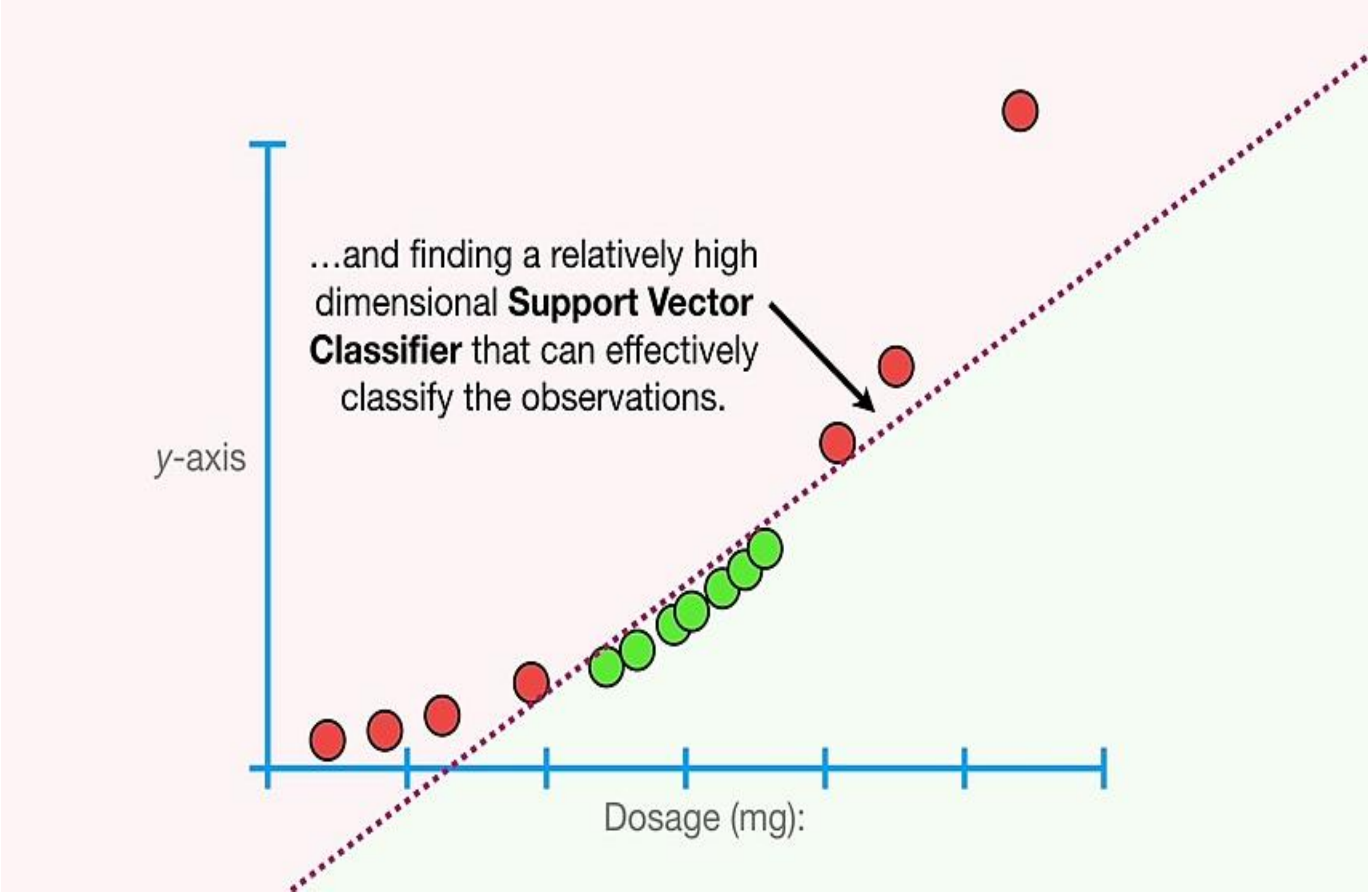
...and it makes calculating relationships in the infinite dimensions used by the **Radial Kernel** possible.
y-axis
Dosage (mg):

However, regardless of how the relationships are calculated, the concepts are the same.

When we have **2** categories, but no obvious linear classifier that separates them in a nice way...

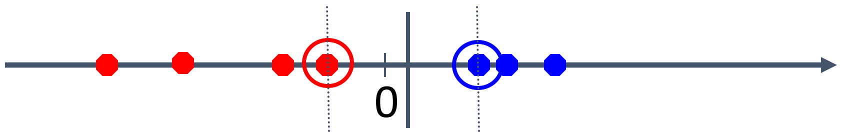



...Support Vector Machines
work by moving the data into a
relatively high dimensional
space...
y-axis
Dosage (mg):


...and finding a relatively high dimensional **Support Vector Classifier** that can effectively classify the observations.
y-axis
Dosage (mg):

Non-linear SVM

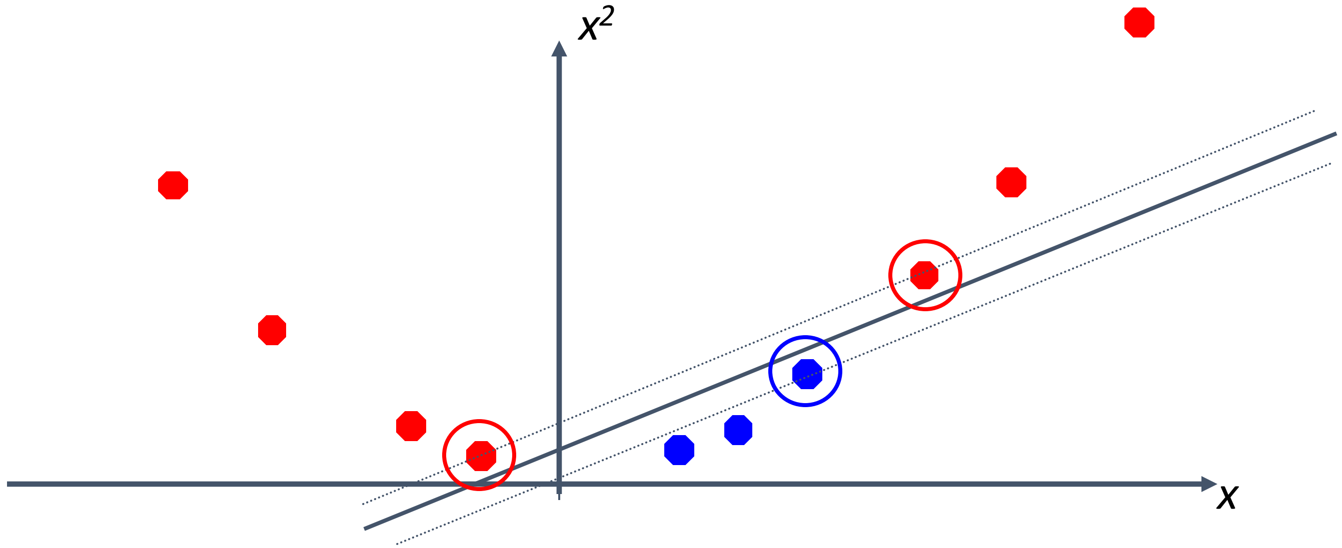
Linearly separable



- What if the decision boundary is not linear?

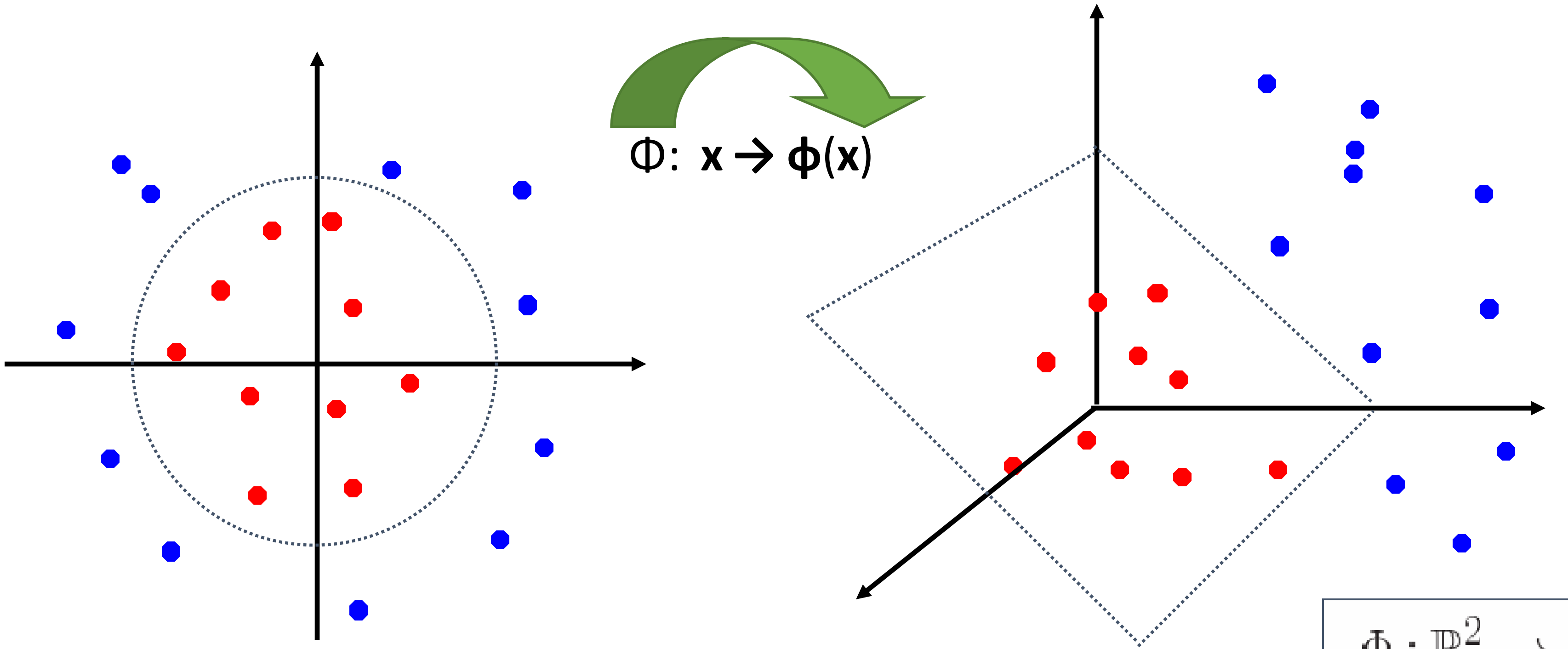
Non-linearly separable

- How about… mapping data to a higher-dimensional space:



Non-linear SVMs: Feature Spaces

Idea: the original feature space can always be mapped to some higher-dimensional feature space where the training set is separable.



$$\Phi : \mathbb{R}^2 \to \mathbb{R}^3$$
$$(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}\, x_1 x_2, x_2^2)$$

Non-linear SVM

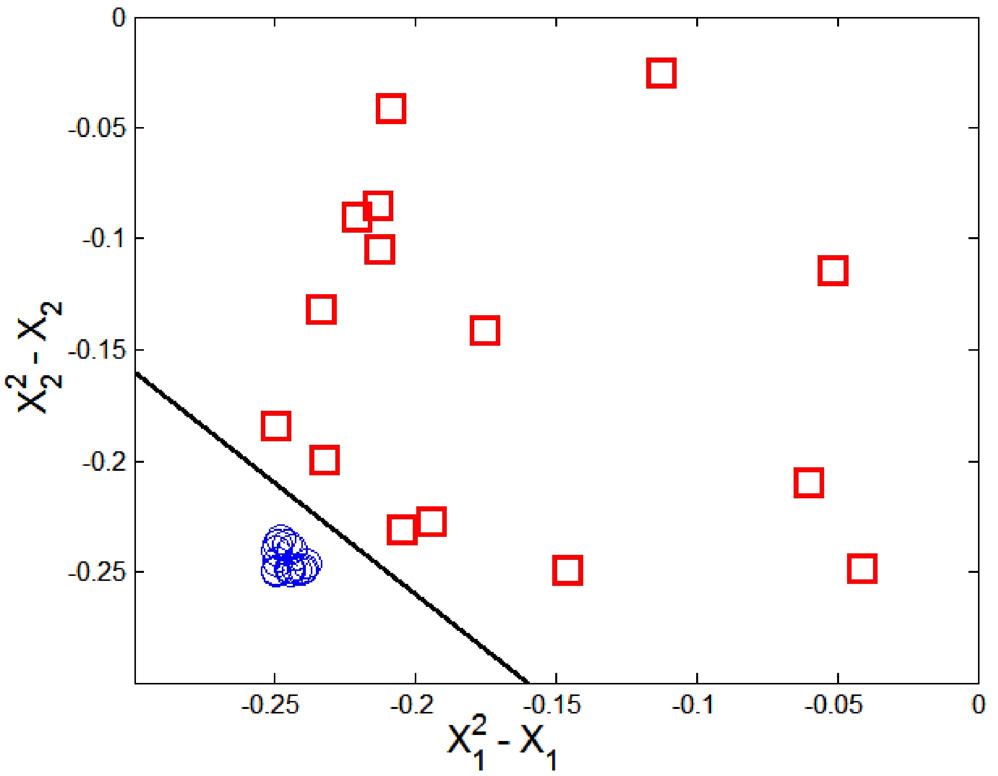
- The trick is to transform the data from its original space x into a new space $\Phi(x)$ (phi) so that a linear decision boundary can be used.

$$x_1^2 - x_1 + x_2^2 - x_2 = -0.46.$$

$$\Phi : (x_1, x_2) \longrightarrow (x_1^2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2, 1).$$

$$w_4 x_1^2 + w_3 x_2^2 + w_2\sqrt{2}x_1 + w_1\sqrt{2}x_2 + w_0 = 0.$$

- Decision boundary $\vec{w} \bullet \Phi(\vec{x}) + b = 0$



Learning a Nonlinear SVM

- Optimization problem

$$\min_{w} \frac{\|\mathbf{w}\|^2}{2}$$

$$\textit{subject to} \quad y_i(w \cdot \Phi(x_i) + b) \geq 1, \ \forall \{(x_i, y_i)\}$$

- Which leads to the same set of equations but involve $\Phi(x)$ instead of x.

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign(\sum_{i=1}^{n} \lambda_i y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{z}) + b).$$

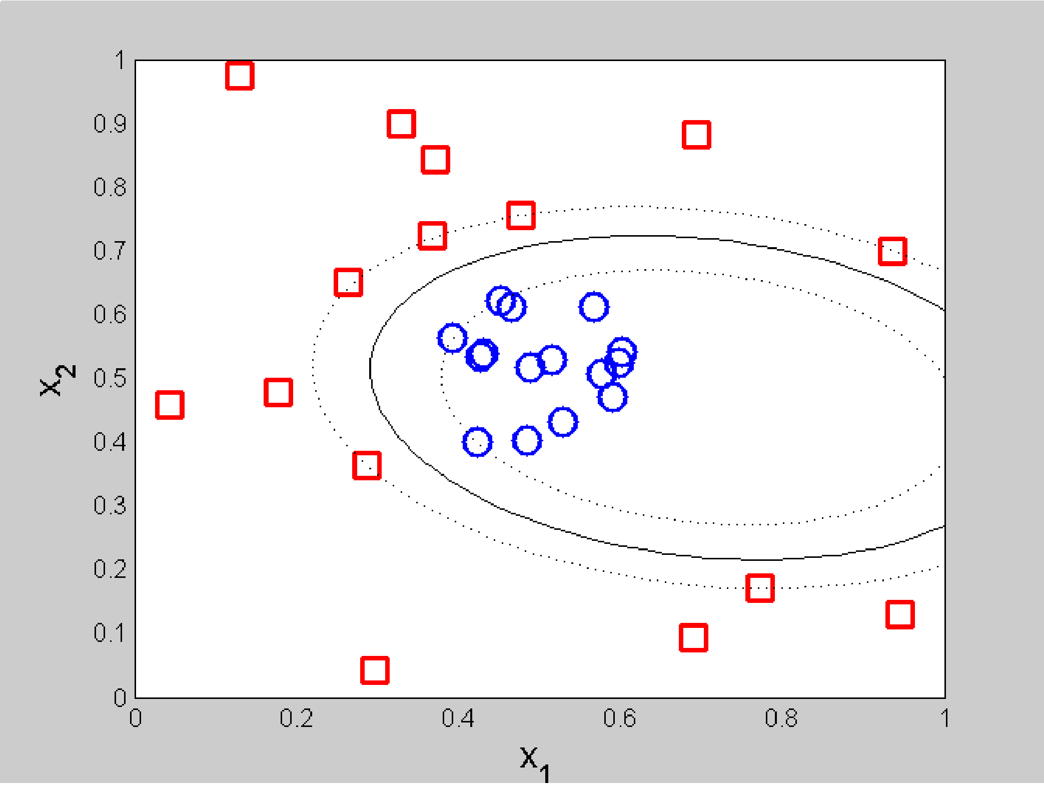
Issues:

- What type of mapping function Φ should be used?
- How to do the computation in high dimensional space?
  - Most computations involve dot product $\Phi(x) \cdot \Phi(x)$
  - Curse of dimensionality?

The Kernel Trick

- $\Phi(x) \cdot \Phi(x) = K(x_i, x_j)$
- $K(x_i, x_j)$ is a kernel function (expressed in terms of the coordinates in the original space)
- Examples:

$$K(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$
$$K(\mathbf{x}, \mathbf{y}) = \exp(-||\mathbf{x} - \mathbf{y}||^2/(2\sigma^2))$$
$$K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$$



https://scikit-learn.org/stable/auto_examples/svm/plot_svm_kernels.html#sphx-glr-auto-examples-svm-plot-svm-kernels-py
https://scikit-learn.org/stable/auto_examples/exercises/plot_iris_exercise.html#sphx-glr-auto-examples-exercises-plot-iris-exercise-py

Examples of Kernel Functions

- Polynomial kernel with degree *d*

$$K(\mathbf{x},\mathbf{y}) = (\mathbf{x}^T\mathbf{y} + 1)^d$$

- Radial basis function kernel with width σ

$$K(\mathbf{x},\mathbf{y}) = \exp(-||\mathbf{x}-\mathbf{y}||^2/(2\sigma^2))$$

  - Closely related to radial basis function neural networks
  - The feature space is infinite-dimensional
- Sigmoid with parameter κ and θ $K(\mathbf{x},\mathbf{y}) = \tanh(\kappa\mathbf{x}^T\mathbf{y} + \theta)$
  - It does not satisfy the Mercer condition on all κ and θ
- Choosing the Kernel Function is probably the most tricky part of using SVM.

The Kernel Trick

- The linear classifier relies on inner product between vectors $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

$$K(\mathbf{x}_i,\mathbf{x}_j)= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$$

- A *kernel function* is a function that is equivalent to an inner product in some feature space.
- Example:

  2-dimensional vectors $\mathbf{x}=[x_1\ \ x_2]$; let $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2$,

  Need to show that $K(\mathbf{x}_i,\mathbf{x}_\mathbf{j})= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$:

  $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j)^2= 1+ x_{i1}^2x_{j1}^2 + 2\, x_{i1}x_{j1}\, x_{i2}x_{j2}+ x_{i2}^2x_{j2}^2 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2}=$

  $= [1\ \ x_{i1}^2\ \ \surd 2\, x_{i1}x_{i2}\ \ x_{i2}^2\ \ \surd 2x_{i1}\ \ \surd 2x_{i2}]^{\mathbf{T}}\, [1\ \ x_{j1}^2\ \ \surd 2\, x_{j1}x_{j2}\ \ x_{j2}^2\ \ \surd 2x_{j1}\ \ \surd 2x_{j2}] =$

  $= \boldsymbol{\phi}(\mathbf{x}_i)^{\mathbf{T}}\boldsymbol{\phi}(\mathbf{x}_j)$, where $\boldsymbol{\phi}(\mathbf{x}) = [1,\ x_1^2,\ \surd 2\, x_1x_2,\ x_2^2,\ \surd 2x_1,\ \surd 2x_2]$

- Thus, a kernel function *implicitly* maps data to a high-dimensional space (without the need to compute each $\boldsymbol{\phi}(\mathbf{x})$ explicitly).

The Kernel Trick

$$f(\mathbf{z}) = sign(\mathbf{w} \cdot \Phi(\mathbf{z}) + b) = sign(\sum_{i=1}^{n} \lambda_i y_i \boxed{K(\mathbf{x}_i, \mathbf{z})} + b).$$

Advantages of using kernel:

- Don't have to know the mapping function Φ.
- Computing dot product $\Phi(x) \cdot \Phi(y)$ in the original space avoids curse of dimensionality.

Not all functions can be kernels

- Must make sure there is a corresponding Φ in some high-dimensional space.
- *Mercer's theorem* (see textbook) that ensures that the kernel functions can always be expressed as the dot product in some high dimensional space.

Mercer theorem: the function must be "positive-definite"

This implies that the *n* by *n* kernel matrix, in which the *(i,j)*-th entry is the $K(x_i, x_j)$, is always positive definite

This also means that optimization problem can be solved in polynomial time!

Constrained Optimization Problem with Kernel

$\text{Minimize} \| \mathbf{w} \| = \langle \mathbf{w} \cdot \mathbf{w} \rangle$ subject to $y_i(\langle \mathbf{x}_i \cdot \mathbf{w} \rangle + b) \geq 1$ for all i

Lagrangian method : maximize $\inf_{\mathbf{w}} L(\mathbf{w}, b, \alpha)$, where

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \| \mathbf{w} \|^2 - \sum_i \alpha_i \left[\left(y_i (\mathbf{x}_i \cdot \mathbf{w}) + b \right) - 1 \right]$$

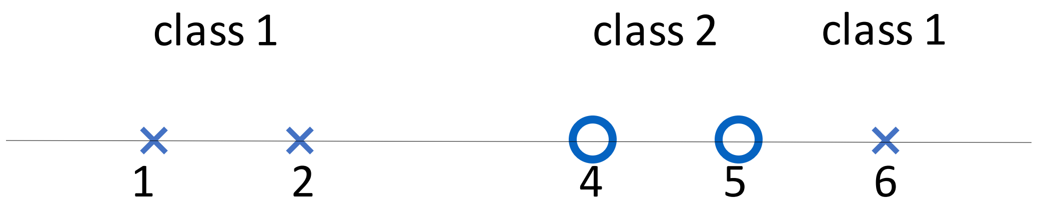
At the extremum, the partial derivative of L with respect both $\mathbf{w}$ and b must be 0. Taking the derivatives, setting them to 0, substituting back into L, and simplifying yields:

$$\text{Maximize} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \, K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to $\sum_i y_i \alpha_i = 0$ and $\alpha_i \geq 0$

$\lambda = \alpha$

Example



Example

- Suppose we have 5 one-dimensional data points
  - x_1=1, x_2=2, x_3=4, x_4=5, x_5=6, with values 1, 2, 6 as class 1 and 4, 5 as class 2
  - $\Rightarrow$ y_1=1, y_2=1, y_3=-1, y_4=-1, y_5=1
- We use the polynomial kernel of degree 2
  - **$K(x,z) = (xz+1)^2$**
  - C is set to 100
- We first find α_i (i=1, …, 5) by

$$\text{max.} \quad \sum_{i=1}^{5} \alpha_i - \frac{1}{2} \sum_{i=1}^{5} \sum_{j=1}^{5} \alpha_i \alpha_j y_i y_j \boxed{(x_i x_j + 1)^2}$$

$$\text{subject to } 100 \geq \alpha_i \geq 0, \sum_{i=1}^{5} \alpha_i y_i = 0$$

Example

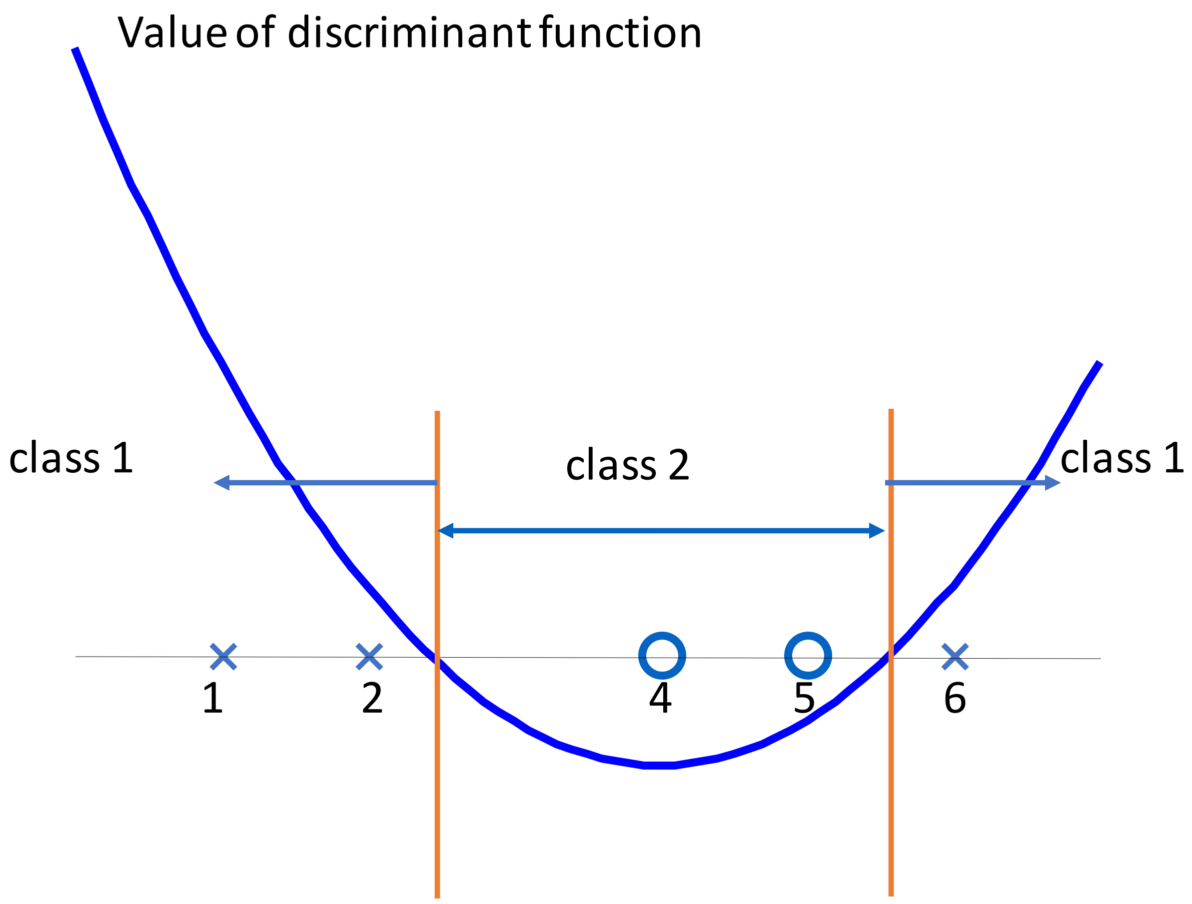
- We get
  - α_1=0, α_2=2.5, α_3=0, α_4=7.333, α_5=4.833
  - Note that the constraints are indeed satisfied
  - The support vectors are {x_2=2, x_4=5, x_5=6}
- The discriminant function is

$$\begin{aligned} & f(z) \\ &= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + \underbrace{4.833}_{\alpha_5}\underbrace{(1)}_{y_5}\underbrace{(6z+1)^2}_{K(z,x_5)} + b \\ &= 0.6667z^2 - 5.333z + b \end{aligned}$$

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x_2 and x_5 lie on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = 1$ and x_4 lies on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = -1$

- All three give b=9 $\Longrightarrow$ $f(z) = 0.6667z^2 - 5.333z + 9$

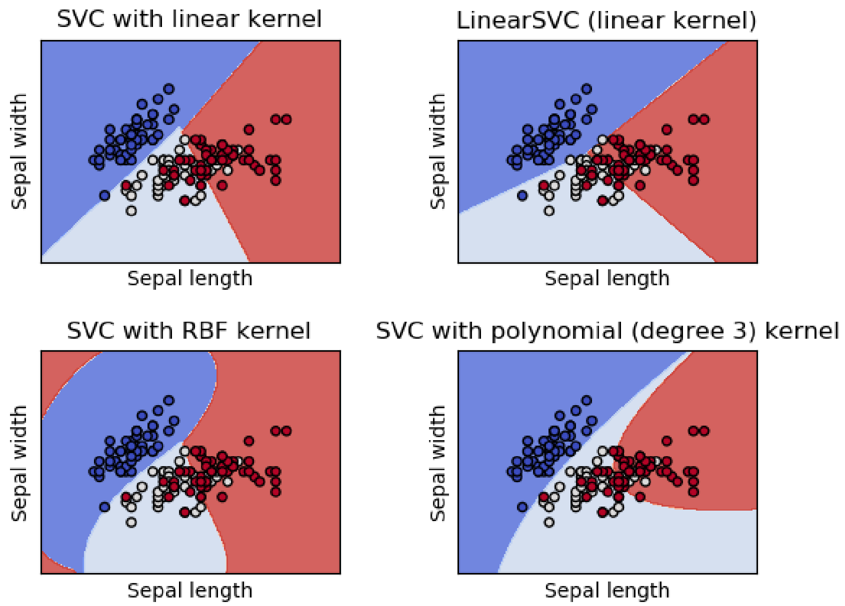
Example



Value of discriminant function
class 1
class 2
class 1
1
2
4
5
6

Support Vector Machine (SVM)

- SVM represents the decision boundary using a subset of the training examples, known as the **support vectors**.
- The basic idea behind SVM lies within the concept of **maximal margin hyperplane.**



Characteristics of SVM

- Since the learning problem is formulated as a convex optimization problem, efficient algorithms are available to find the **global** minima of the objective function (many of the other methods use greedy approaches and find **locally** optimal solutions).
- Overfitting is addressed by maximizing the margin of the decision boundary, but the user still needs to provide the type of kernel function and cost function.
- Difficult to handle missing values.
- Robust to noise.
- High computational complexity for building the model.

References

- Support Vector Machine (SVM). Chapter 5.5. Introduction to Data Mining.
- http://www.kernel-machines.org/
- http://www.support-vector.net/
- An Introduction to Support Vector Machines. N. Cristianini and J. Shawe-Taylor.
- C.J.C. Burges: A tutorial on Support Vector Machines. Data Mining and Knowledge Discovery 2:121-167, 1998.