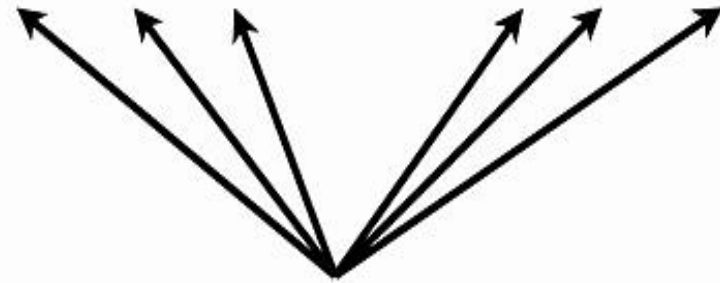# DATA MINING 2
Support Vector Machine

Riccardo Guidotti

a.a. 2023/2024

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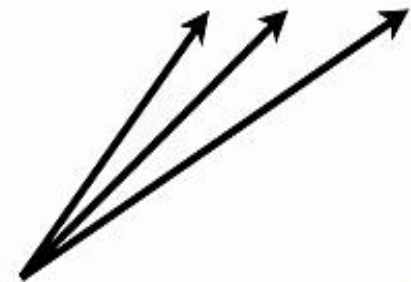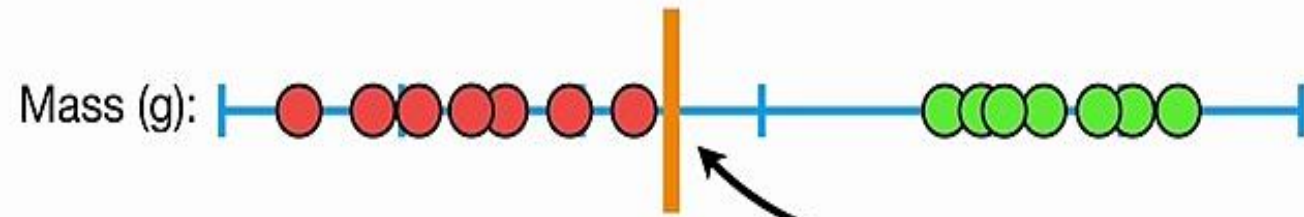Contains slides integrated with StatQuest

UNIVERSITÀ DI PISA

Mass (g):
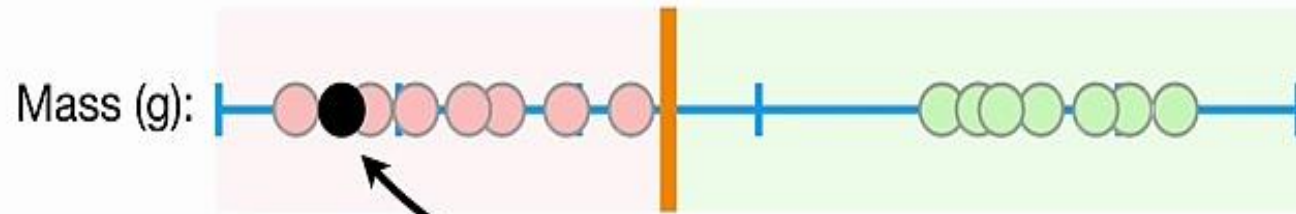
The **red dots** represent mice are *not obese*…

Mass (g):

...and the **green dots** represent mice are *obese*.
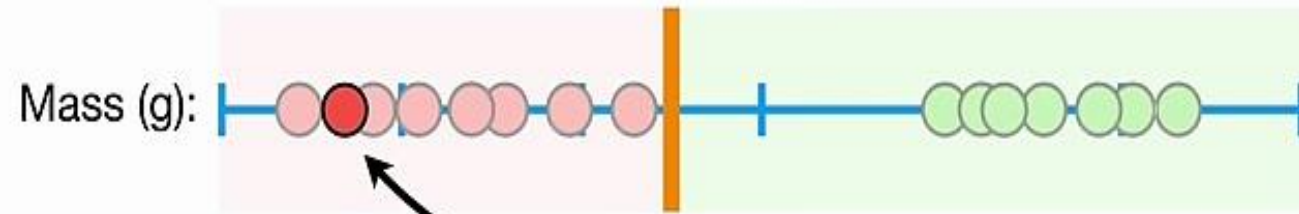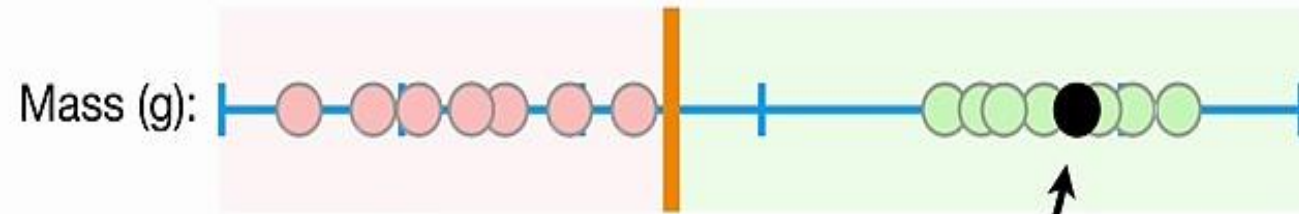
Based on these observations, we can pick a threshold…

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

Mass (g):

…we can classify it as **_not obese_**.
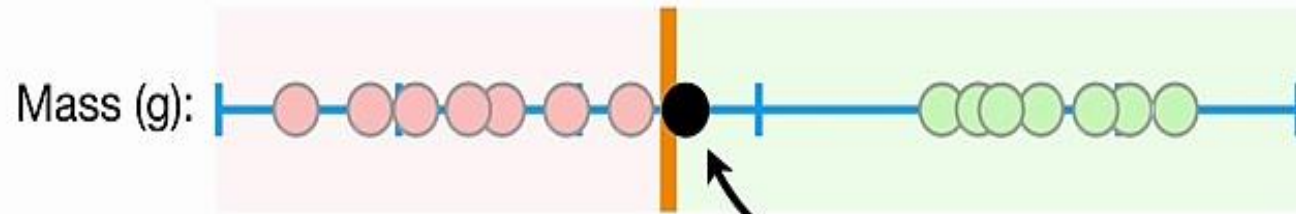
Mass (g):

And when we get a new observation with more mass than the threshold…

Mass (g):
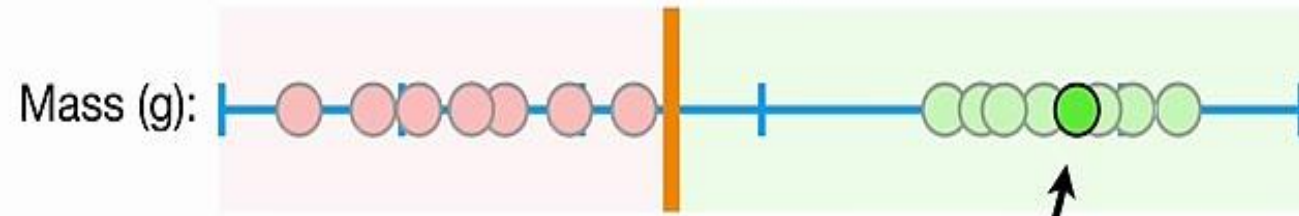
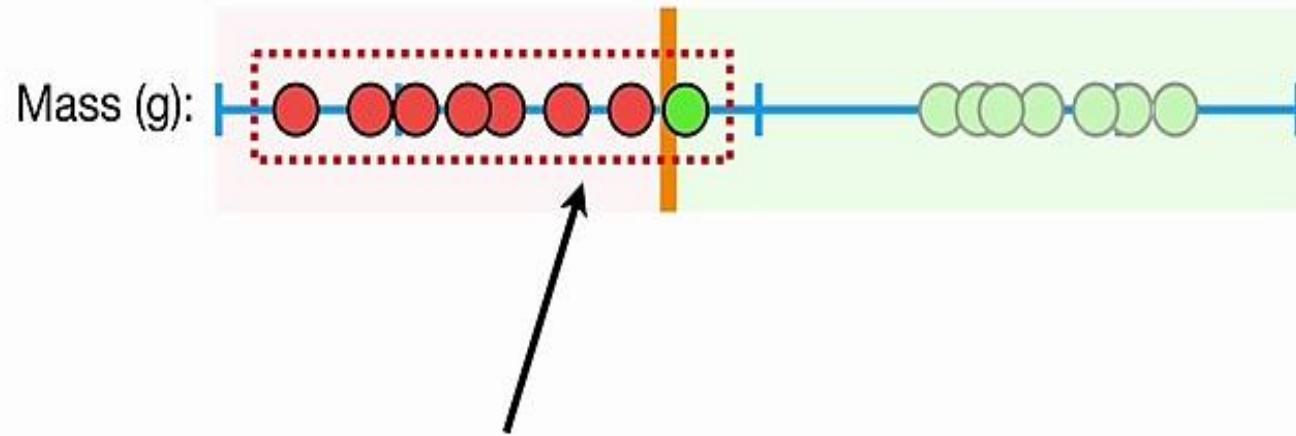...we can classify it as *obese*.

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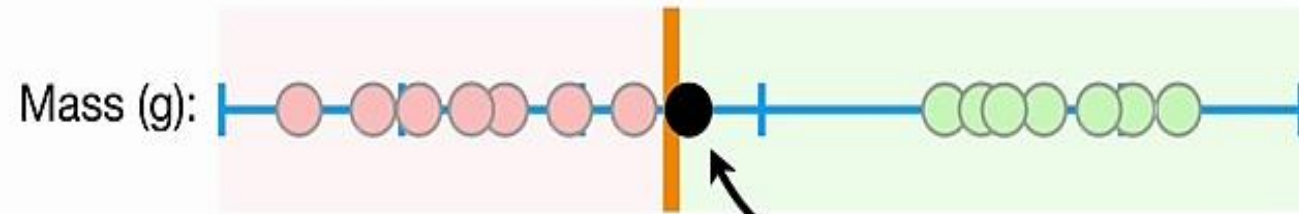
However, what if get a new observation here?

Mass (g):

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

Mass (g):

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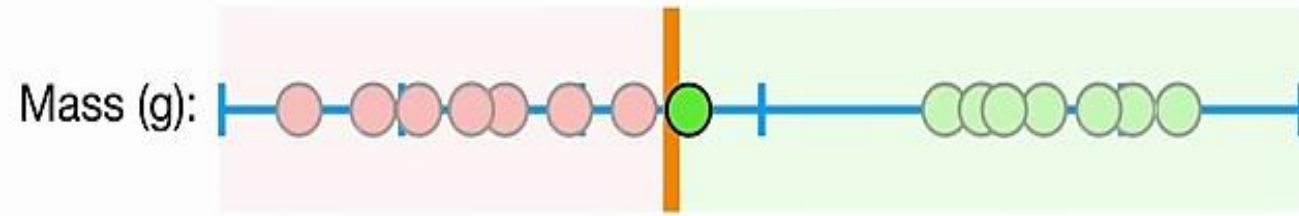
So this threshold is pretty lame.

Mass (g):
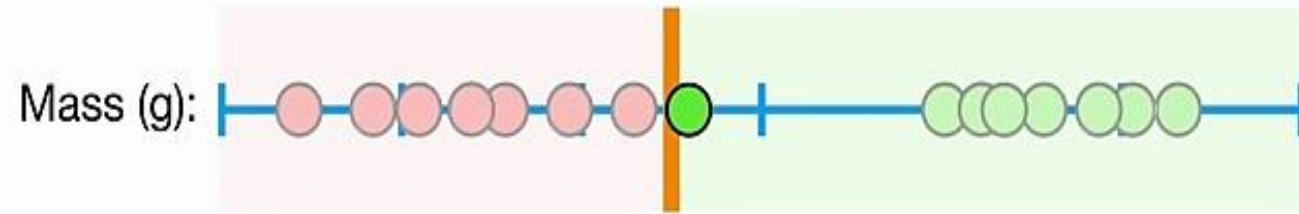
Can we do better?
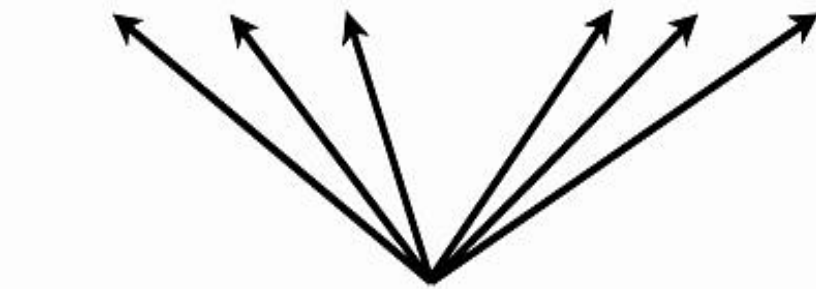
Going back to the original training dataset...

Mass (g):
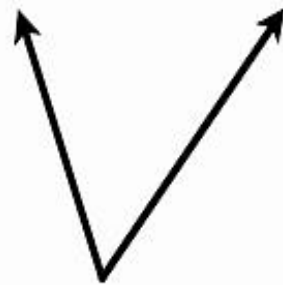
...we can focus on the observations on
the edges of each cluster...
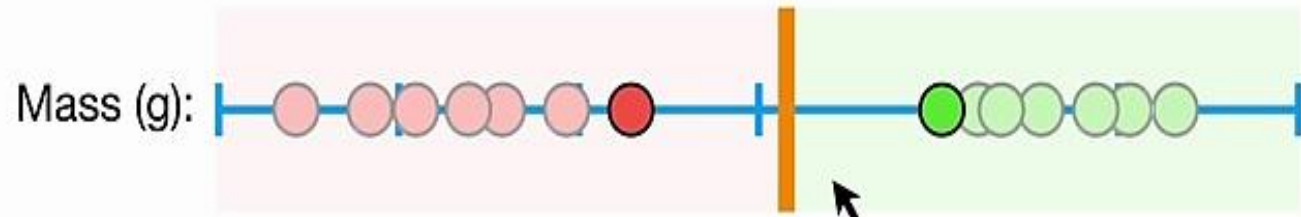
...and use the midpoint between them as the threshold.

Mass (g):
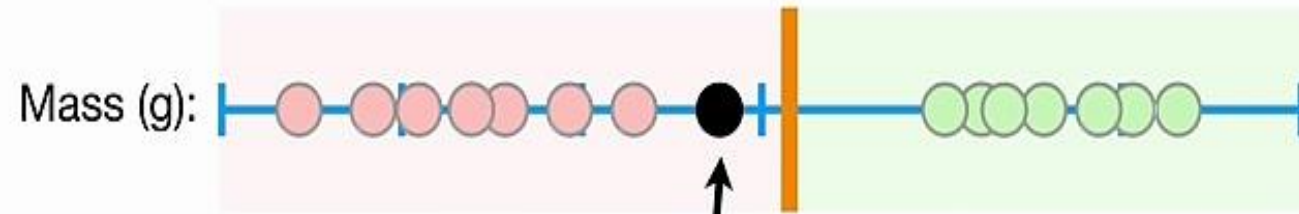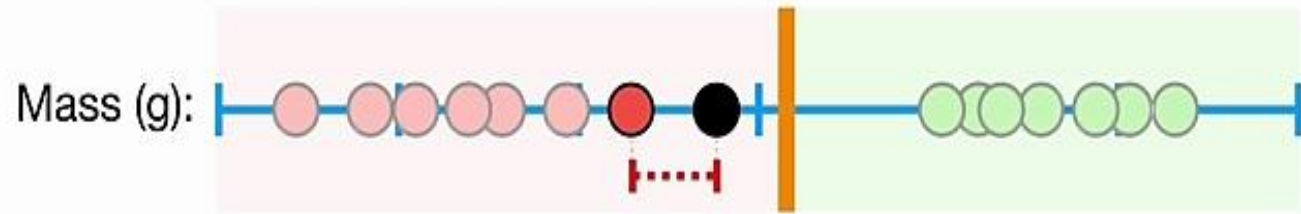
...it will be closer to the
observations that are ***not obese***...

Mass (g):
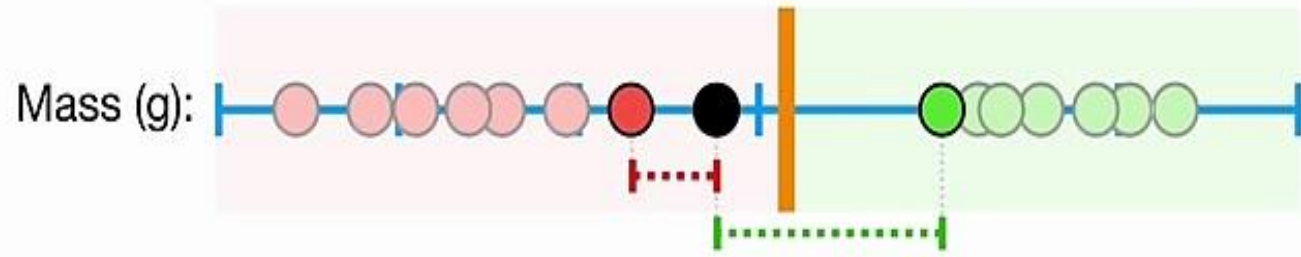
...than it is to the *obese* observations.

Mass (g):
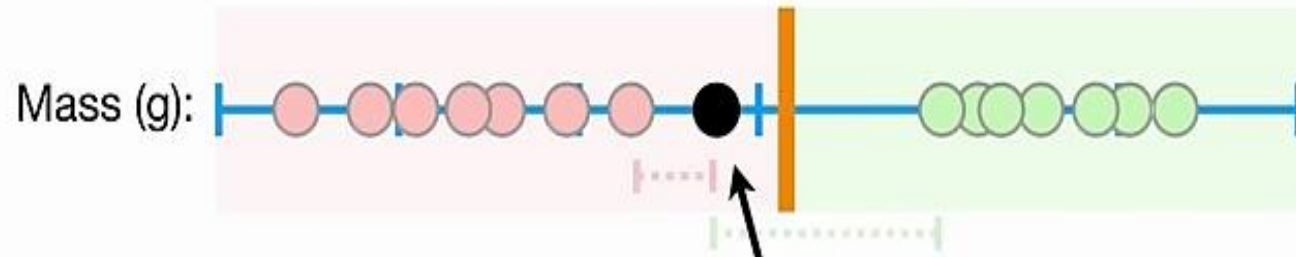
So it makes sense to classify this new observation as *not obese*.

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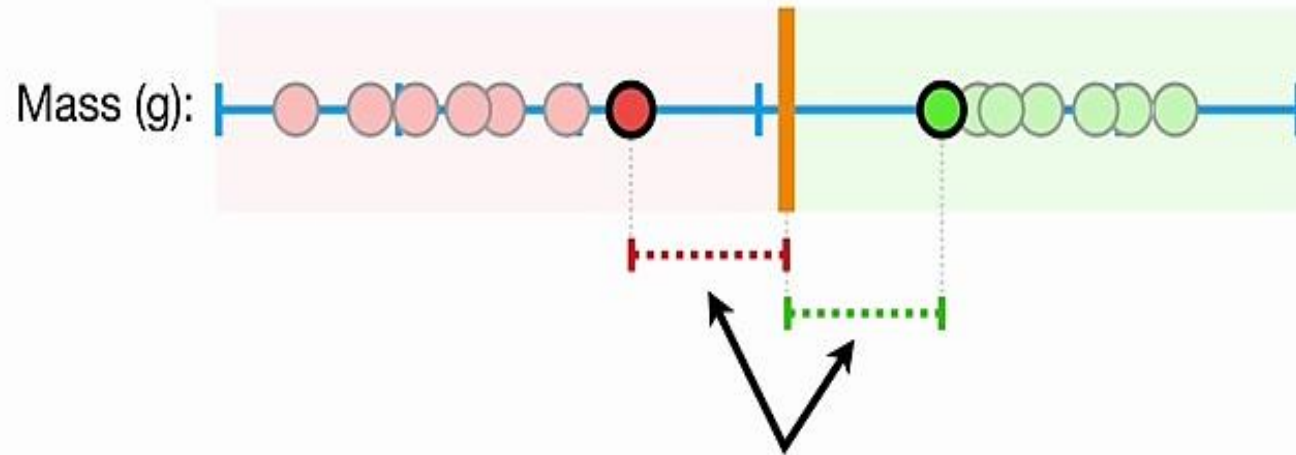
The shortest distance between the observations and the threshold is called the **margin**.

Mass (g):
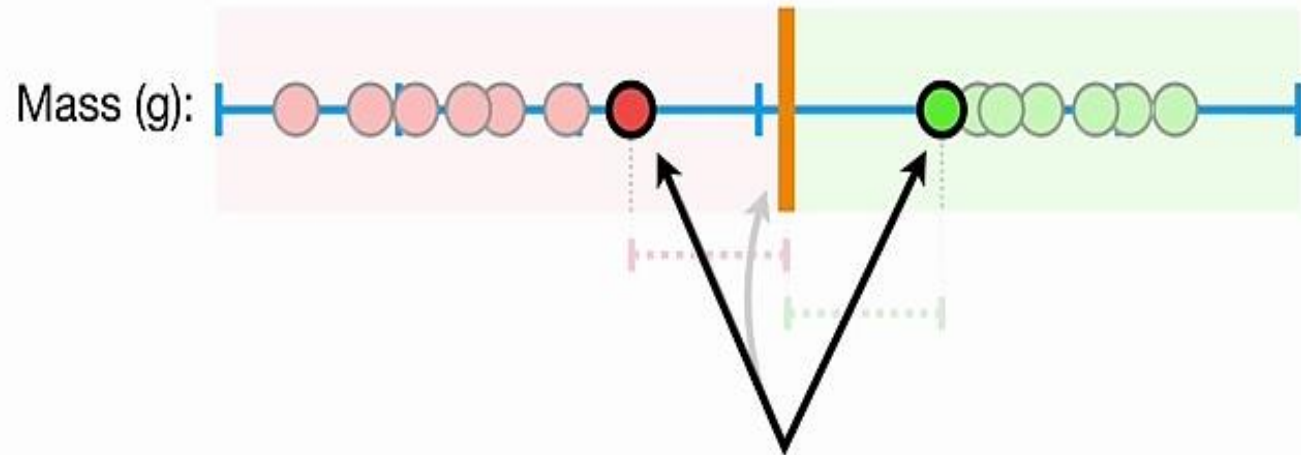
Since we put the threshold halfway between these two observations...

Mass (g):
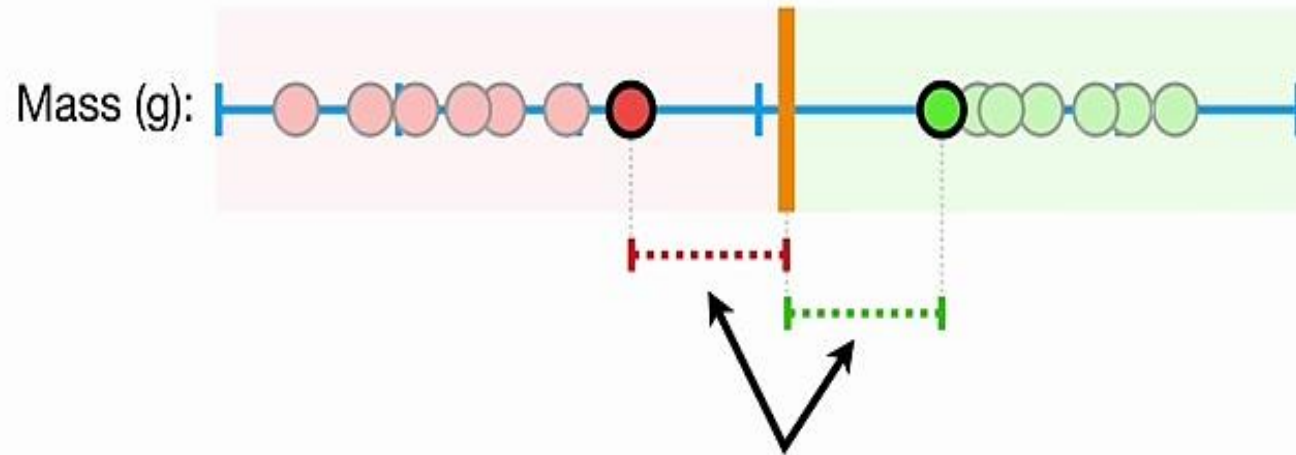
…the distances between the observations and the threshold are the same and both reflect the **margin**.

Mass (g):
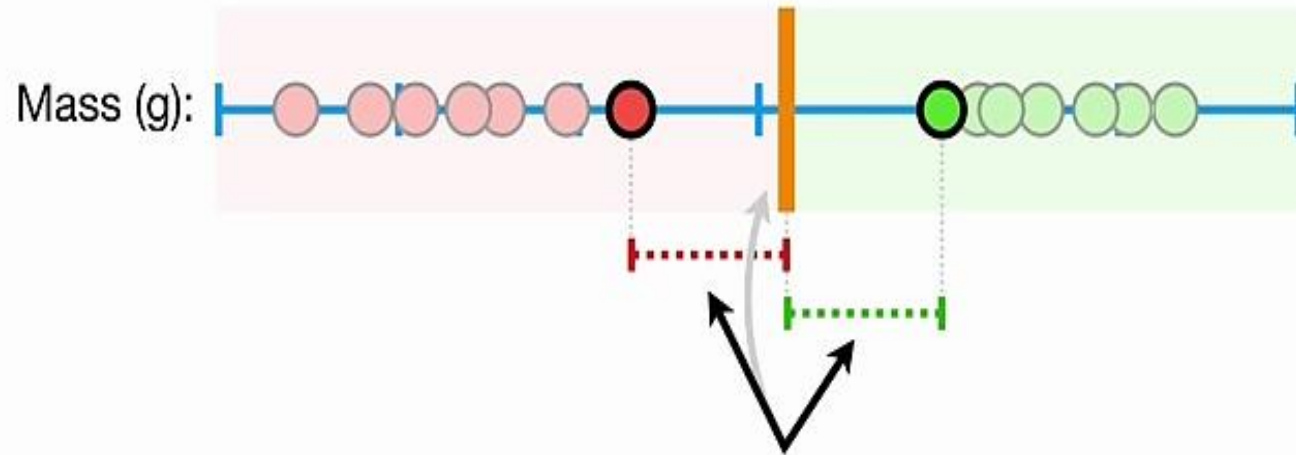
When the threshold is halfway between the two observations, the **margin** is as large as it can be.
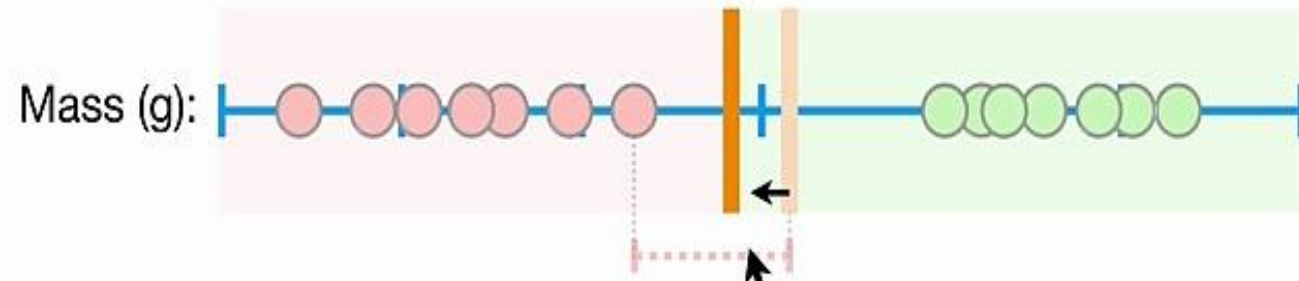
Mass (g):

For example, if we moved the
threshold to the left a little bit...
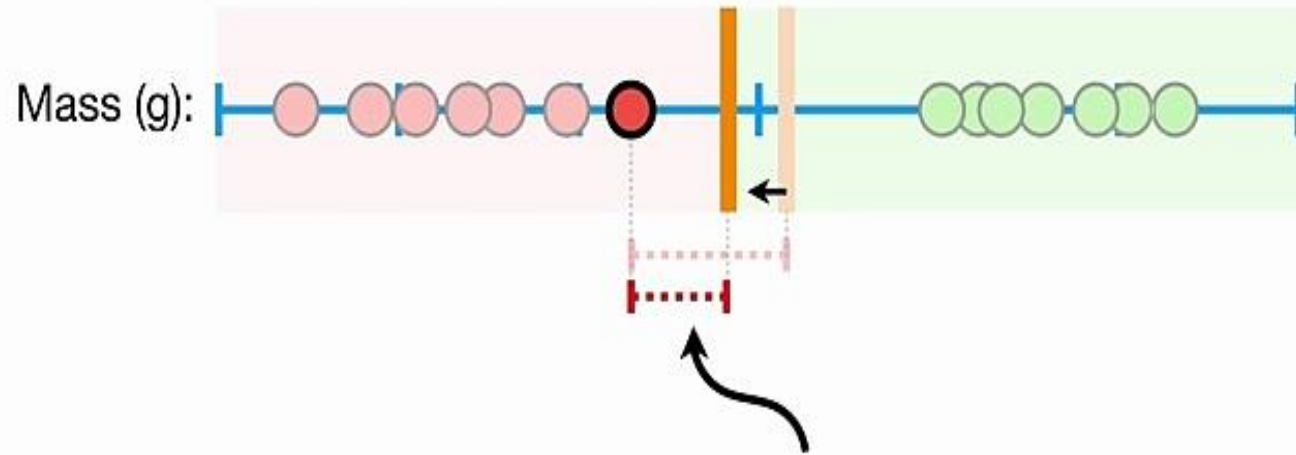
Mass (g):

...then the distance between the threshold and the observation that is *not obese* would be smaller...
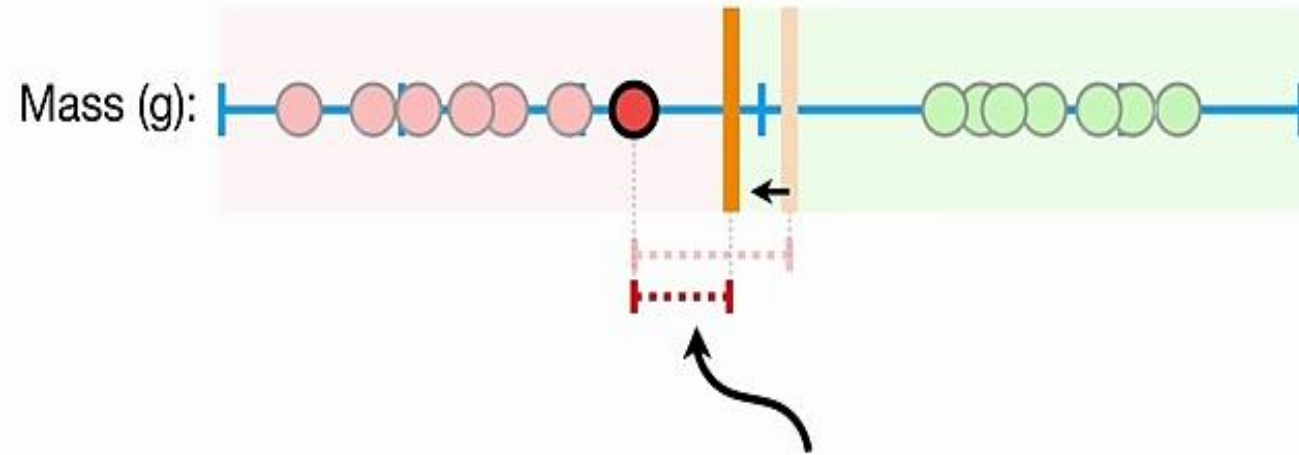
Mass (g):

...and thus, the **margin** would be smaller than it was before.
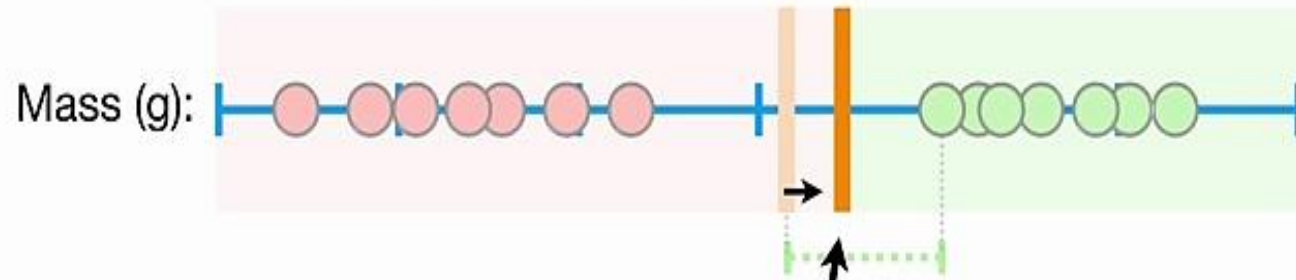
And if we moved the threshold to
the right a little bit…

Mass (g):
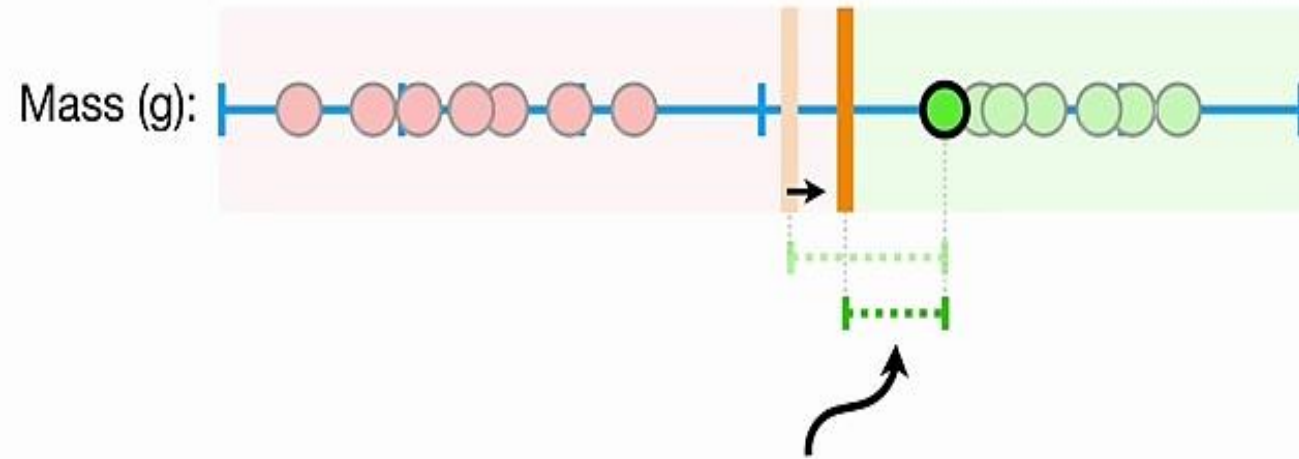
...then the distance between the *obese* observation and the threshold would get smaller...

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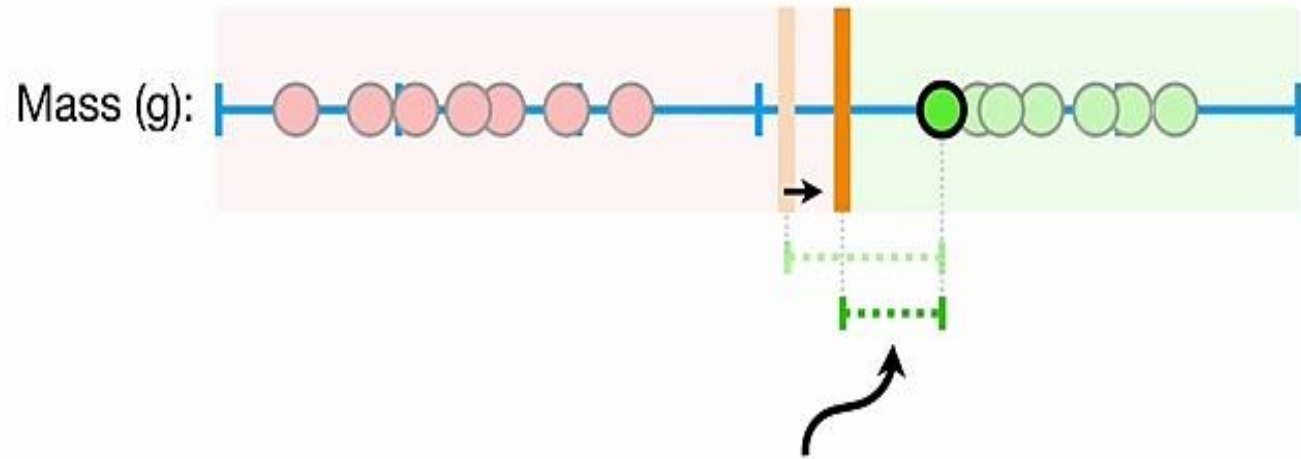
…and again, the **margin** would be smaller.

Mass (g):
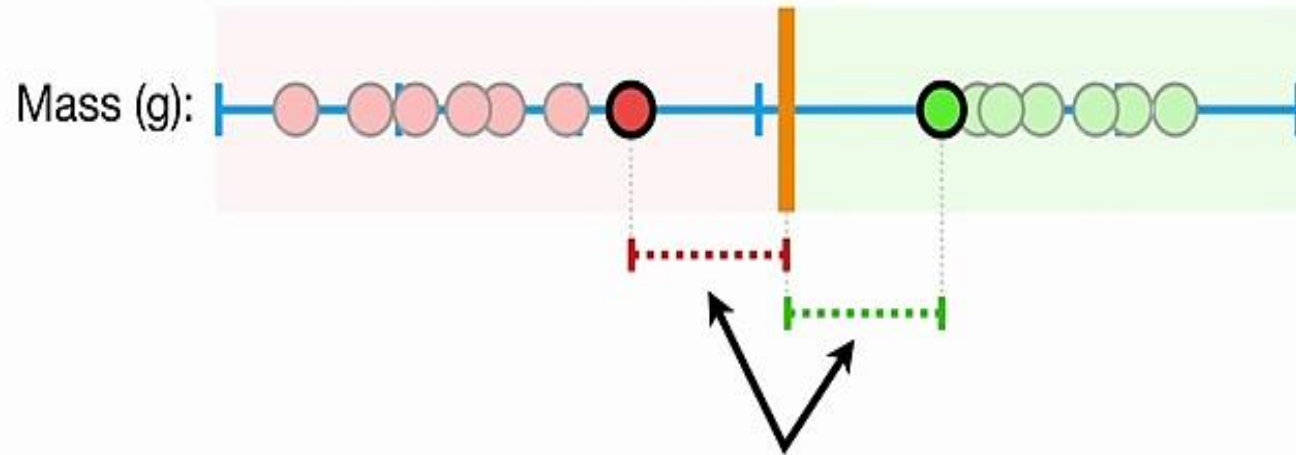
When we use the threshold that gives us the largest **margin** to make classifications…
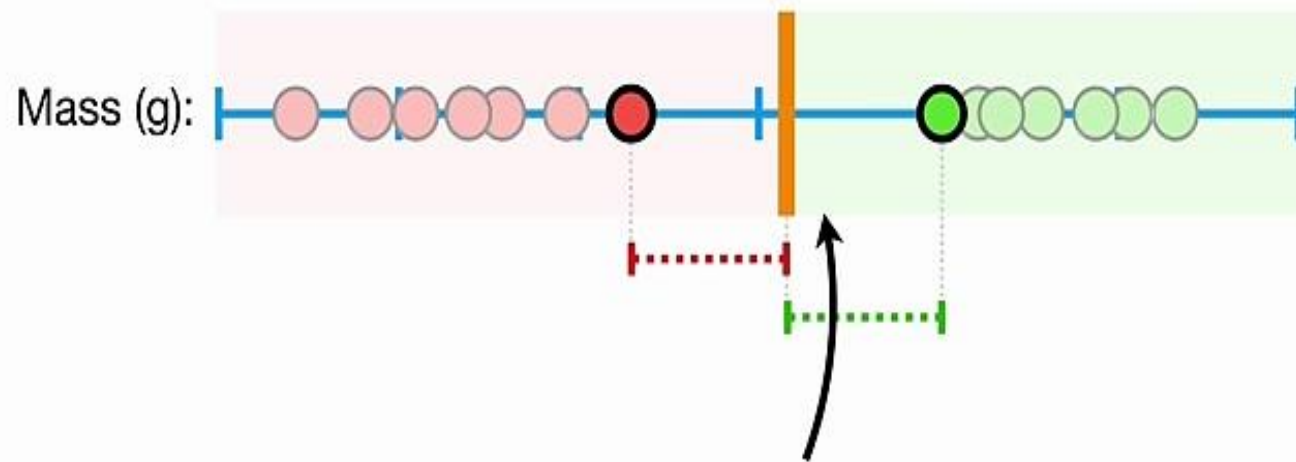
...we are using a **Maximal Margin Classifier**.
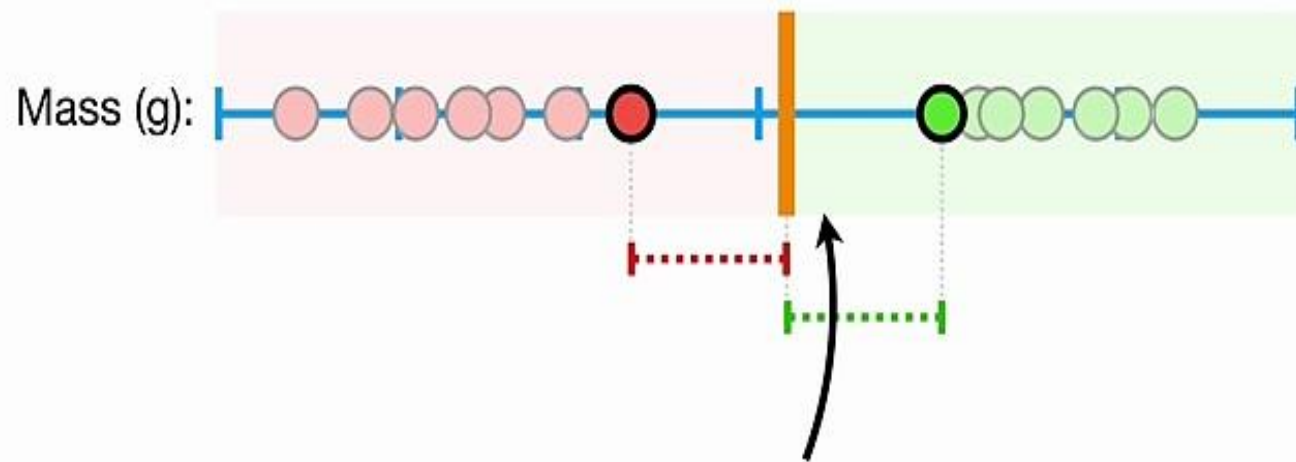
Maximal Margin Classifiers
seem pretty cool...
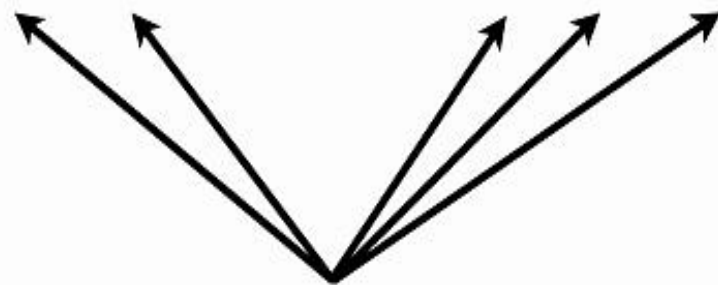
...but what if our training data looked like this....

Mass (g):

…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.
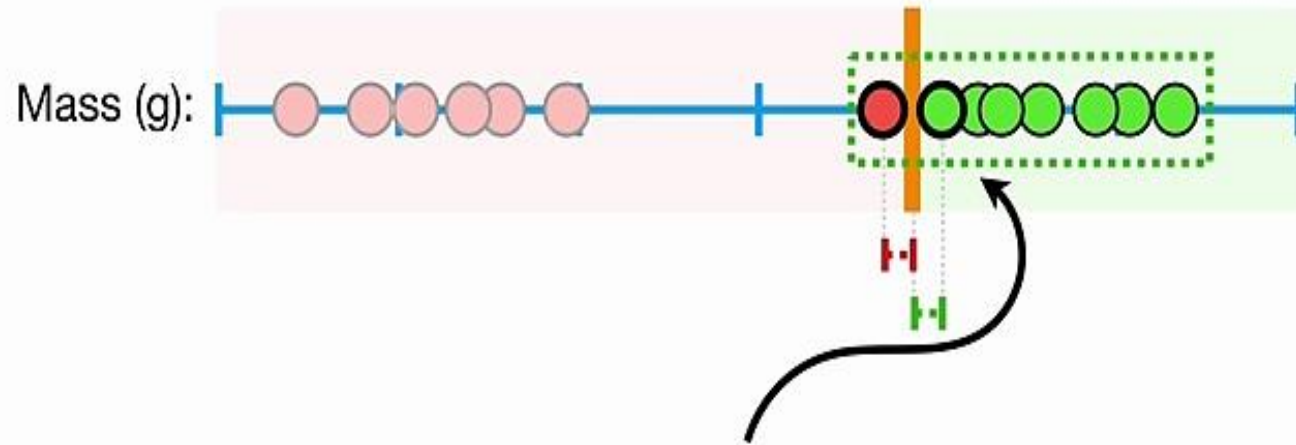
In this case, the **Maximum Margin Classifier** would be super close to the *obese* observations…
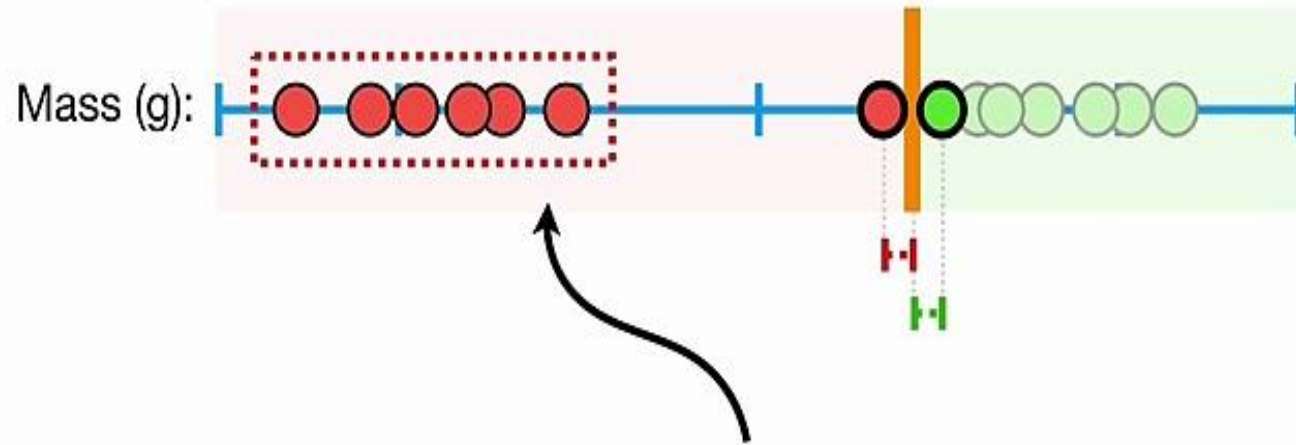
...and really far from the majority of the observations that are **not obese**.
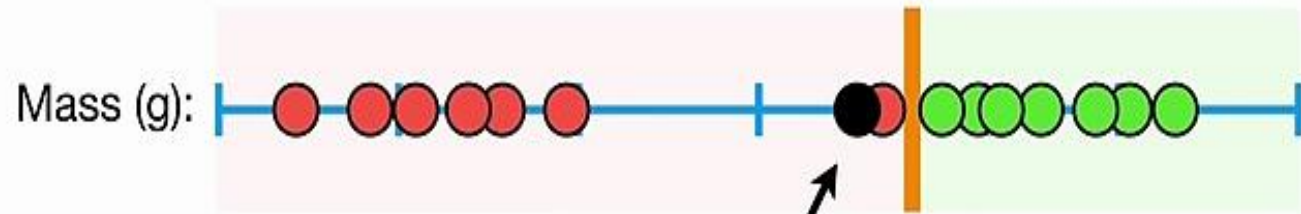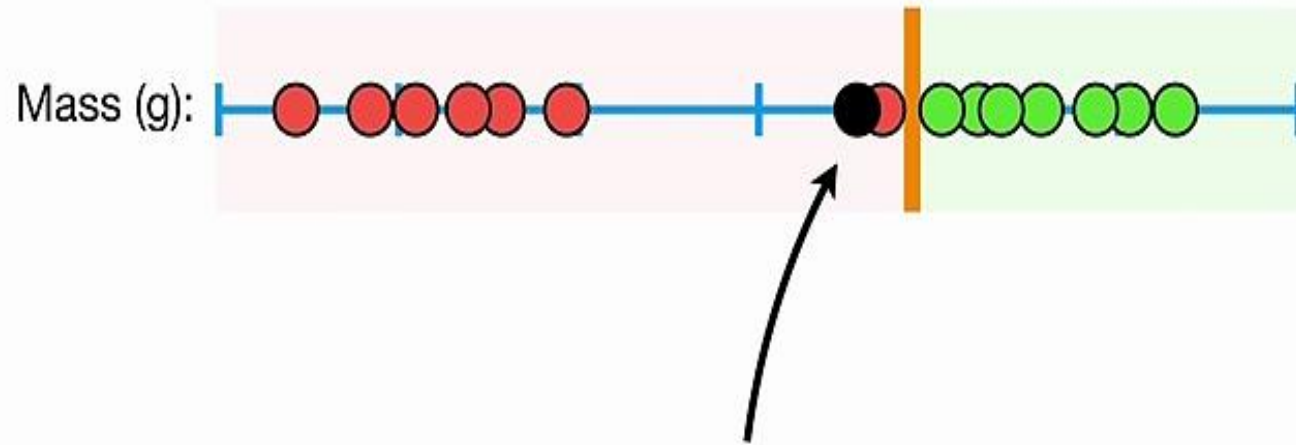
...we would classify it as *not obese*, even though most of the *not obese* observations are much further away than the *obese* observations.
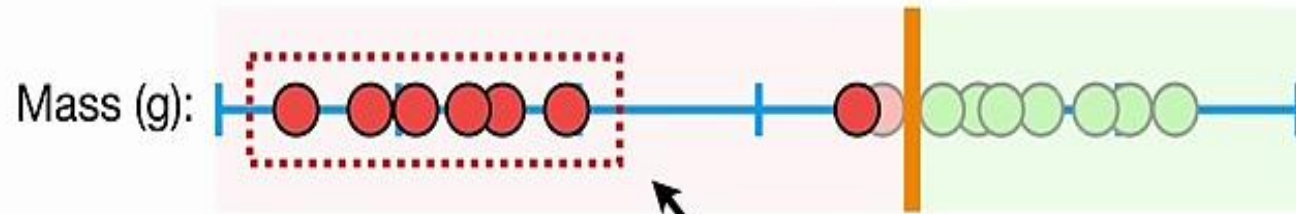
...we would classify it as **not obese**, even though most of the **not obese** observations are much further away than the **obese** observations.
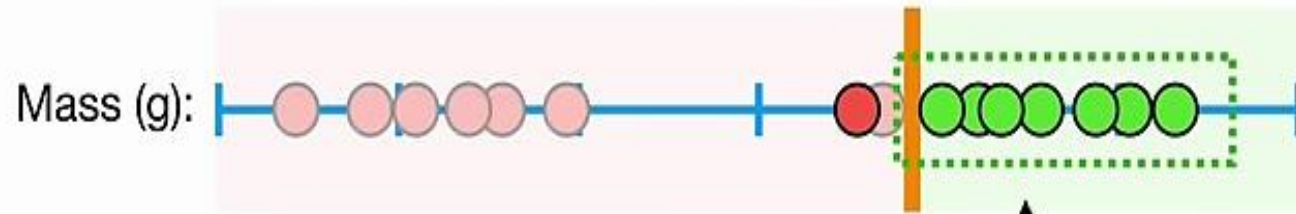
...we would classify it as **_not obese_**, even though most of the **_not obese_** observations are much further away than the *obese* observations.
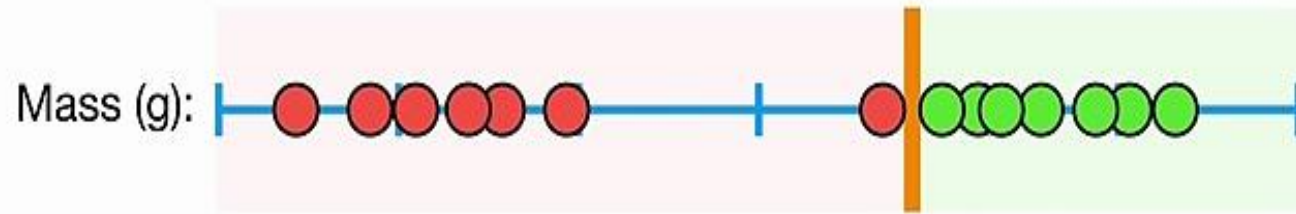
So **Maximal Margin Classifiers** are *super sensitive to outliers* in the training data and that makes them pretty lame.

Mass (g):
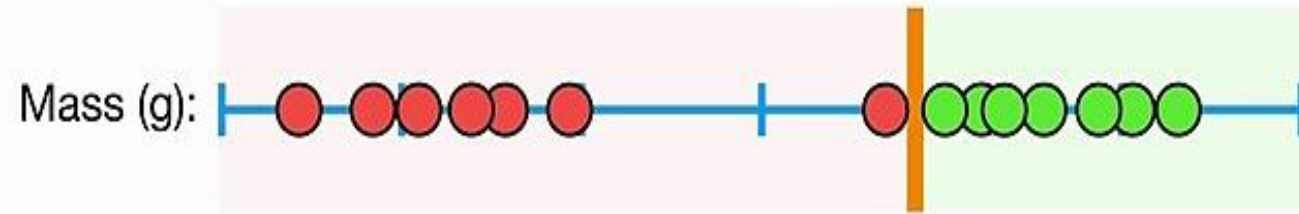
Can we do better?

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

Mass (g):
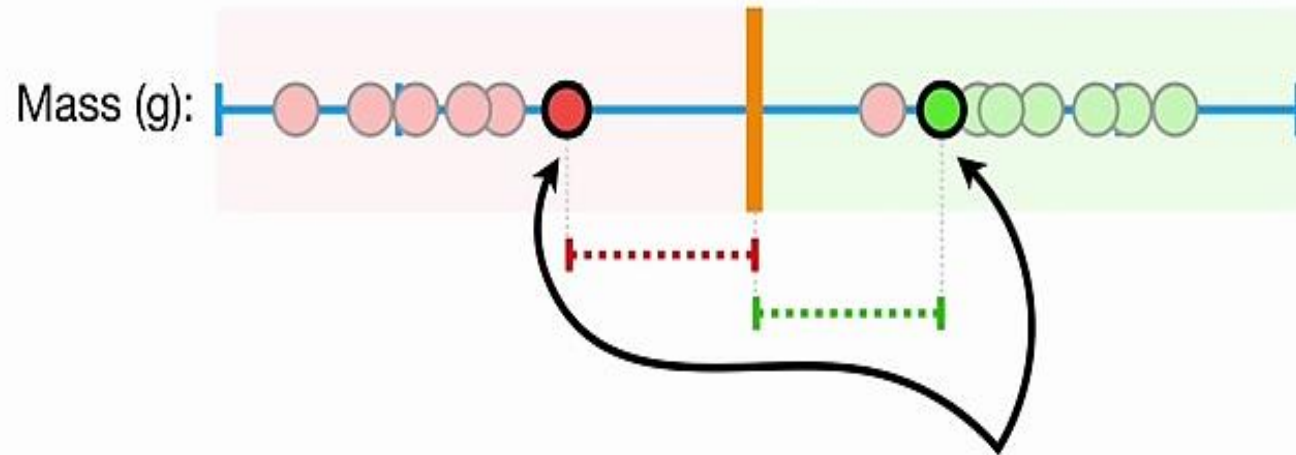
For example, if we put the threshold halfway between these two observations…

Mass (g):
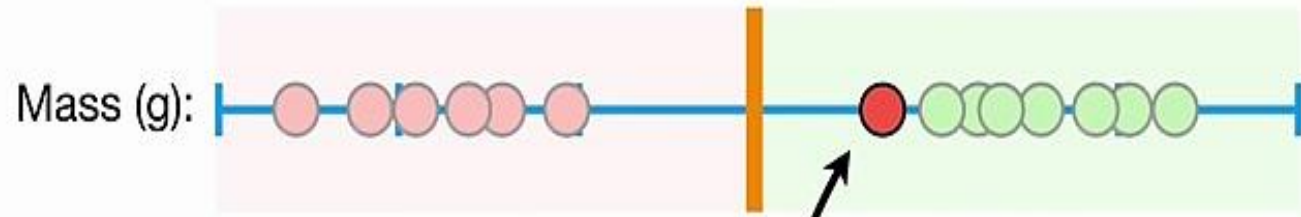
…then we will misclassify this observation.

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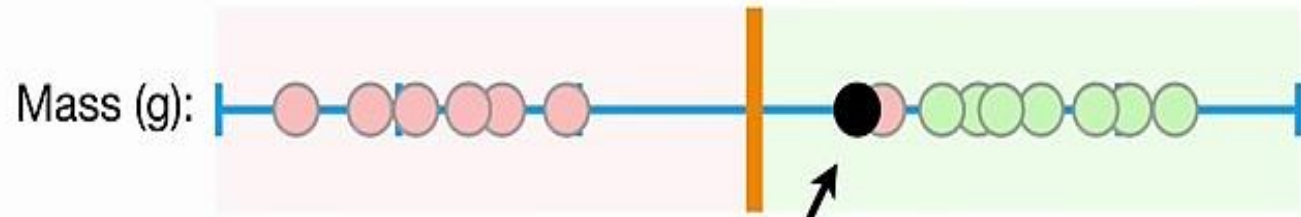
However, now when we get a
new observation here…

Mass (g):
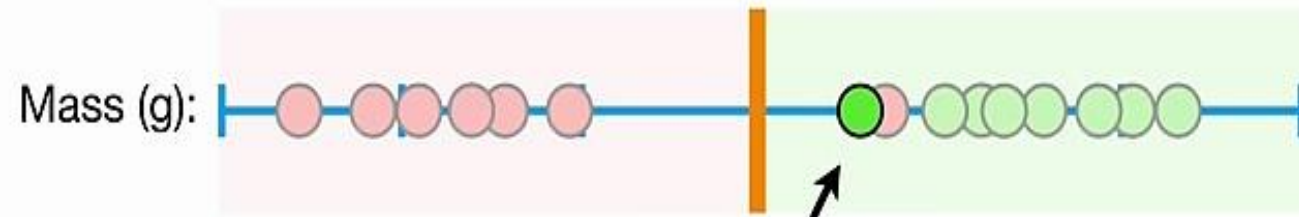
...we will classify it as *obese*...

Mass (g):
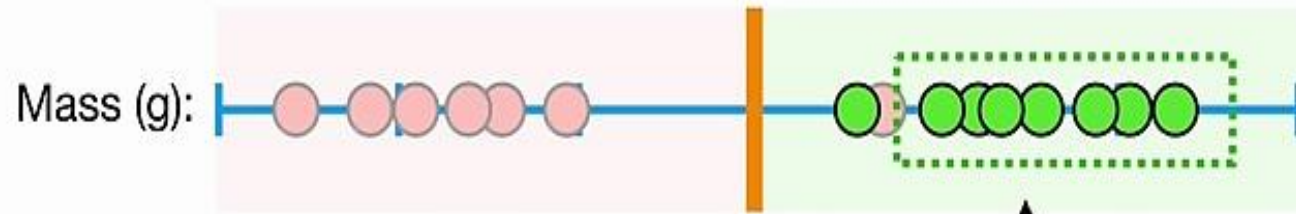
...and that makes sense because it is closer to most of the *obese* observations.

Mass (g):
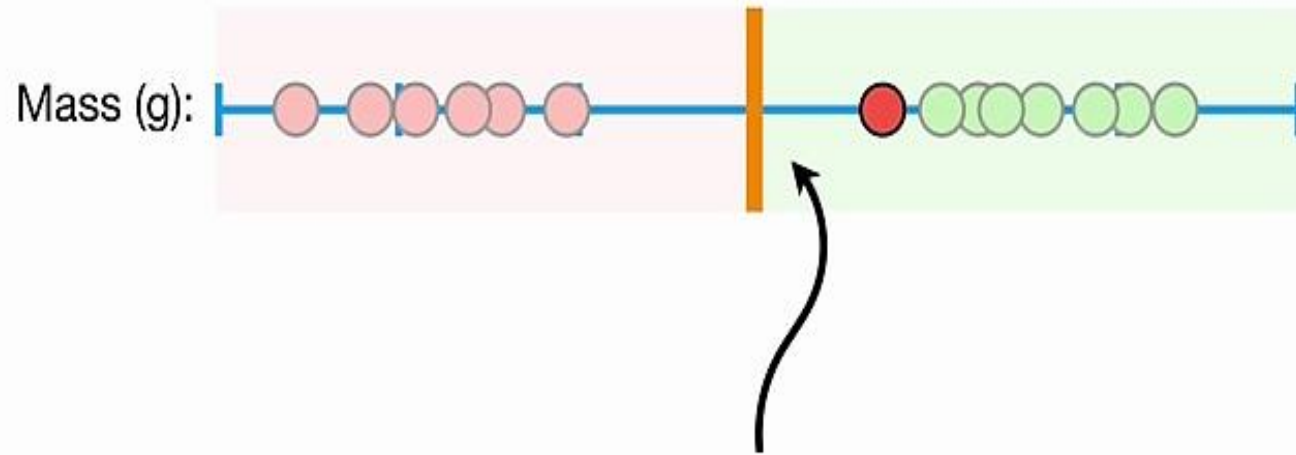
Choosing a threshold that allows
misclassifications is an example of
the **Bias/Variance Tradeoff** that
plagues all of machine learning.
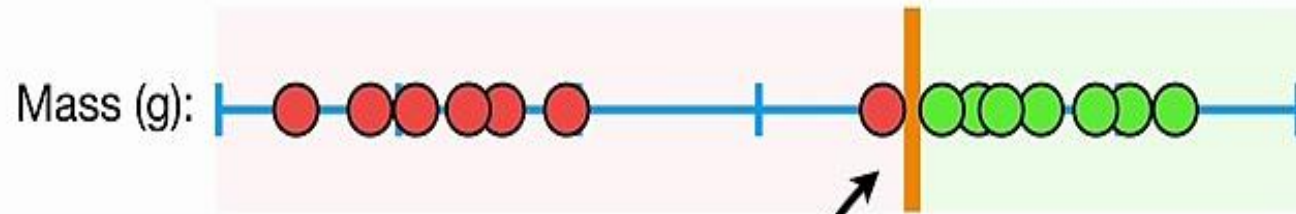
In other words, before we allowed misclassifications, we picked a threshold that was very sensitive to the training data (low bias)…
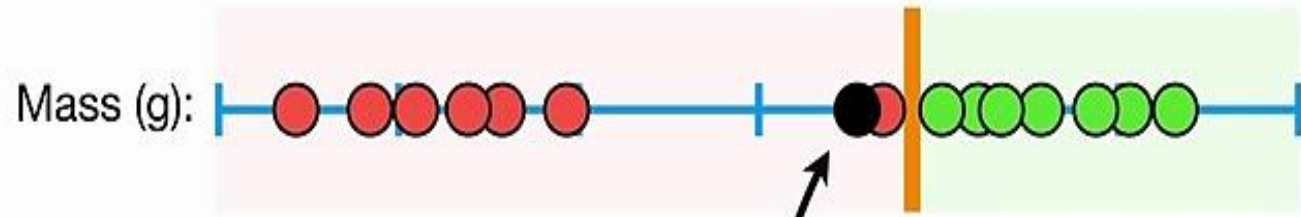
...and it performed poorly when we got new data (high variance).
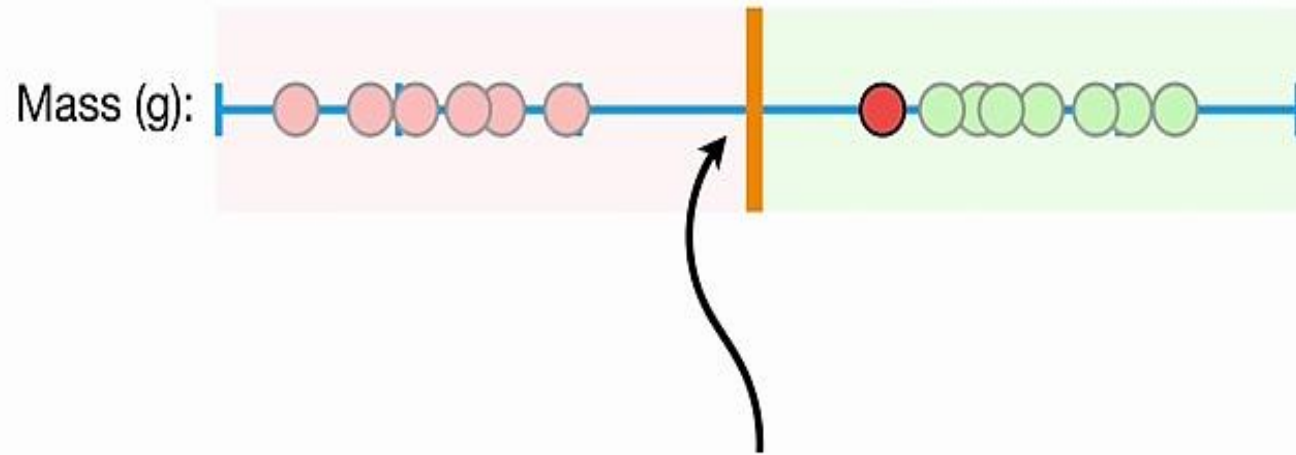
In contrast, when we picked a threshold that was less sensitive to the training data and allowed misclassifications (higher bias)…

Mass (g):
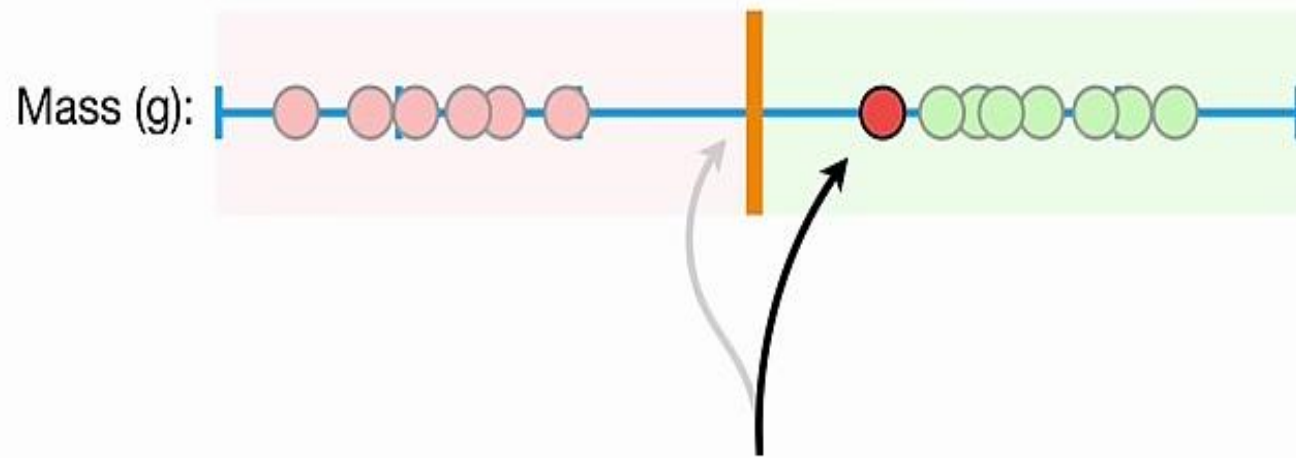
In contrast, when we picked a threshold that was less sensitive to the training data and allowed misclassifications (higher bias)…
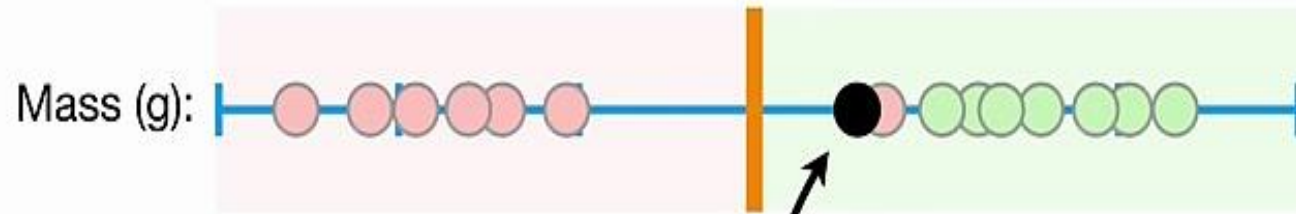
...it performed better when we got new data (low variance).
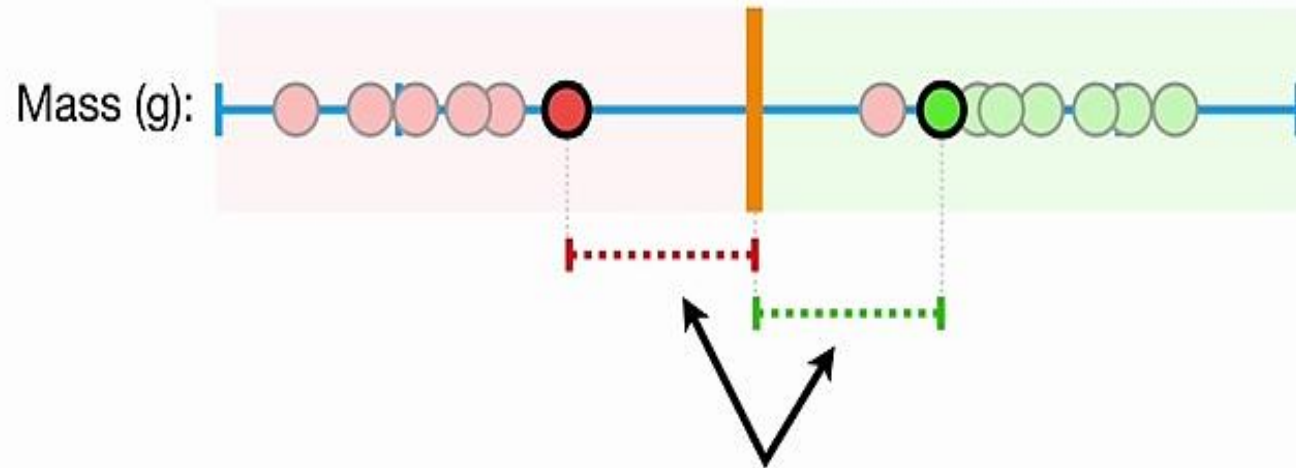
Mass (g):

When we allow misclassifications, the distance between the observations and the threshold is called a **Soft Margin**.
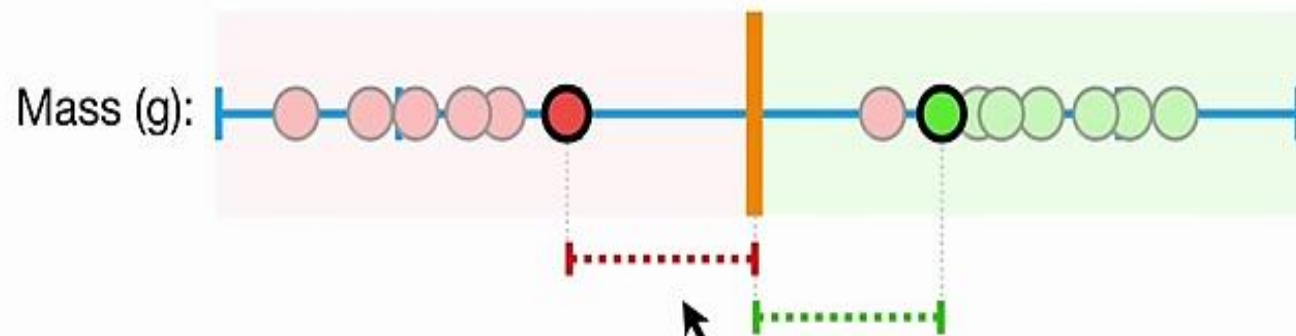
Mass (g):

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

Mass (g):
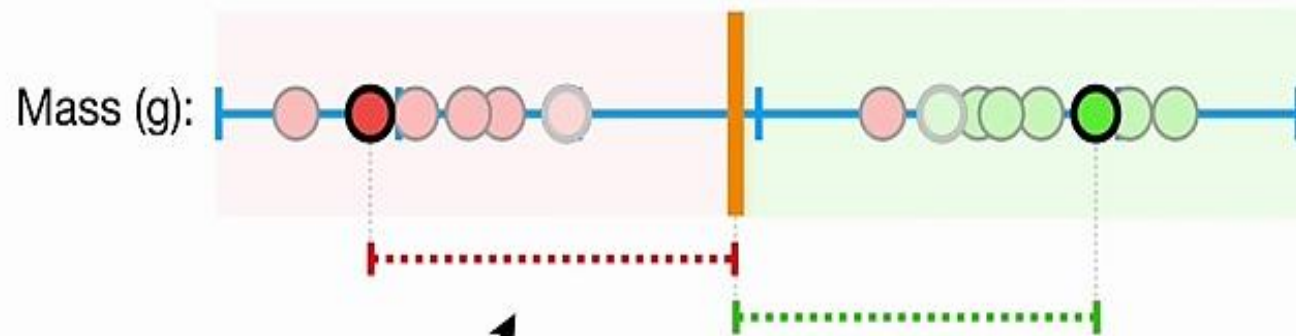
So the question is "How do we know that this **soft margin**…

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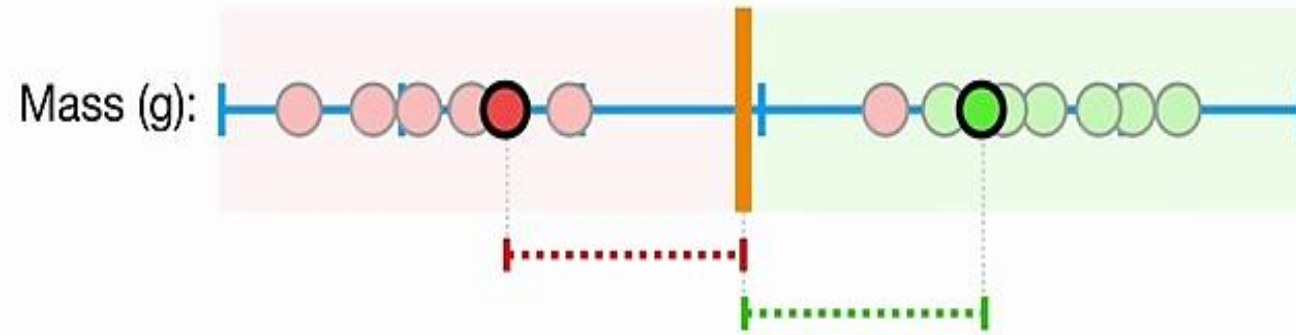
When we use a **Soft Margin** to determine the location of a threshold…

Mass (g):
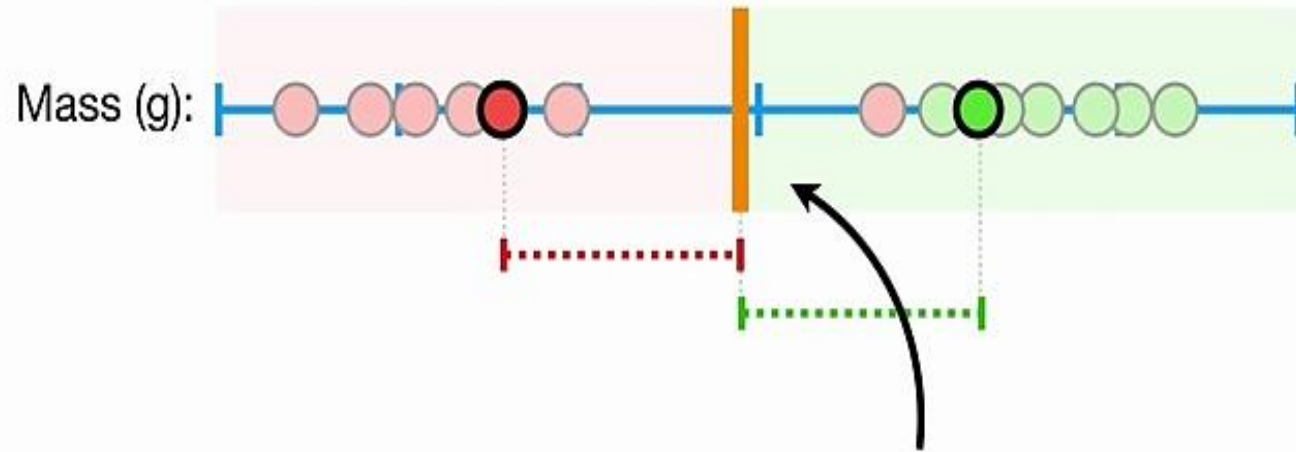
...then we are using a **Soft Margin Classifier** aka a **Support Vector Classifier** to classify observations.
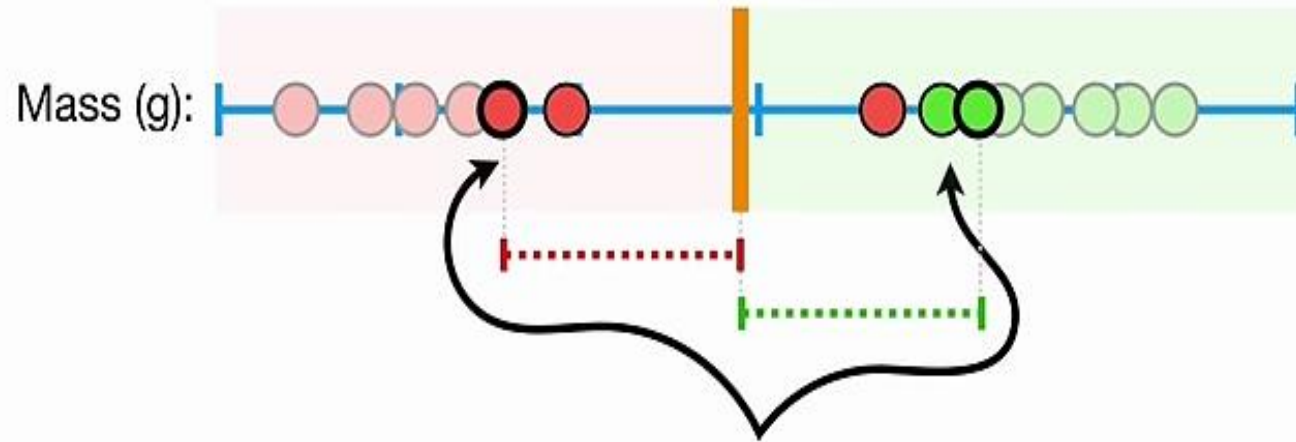
The name **Support Vector Classifier** comes from the fact that the observations on the edge *and within* the **Soft Margin** are called **Support Vectors**.

$$w x + b = 0$$

$$w x + b = -1 \qquad\qquad\qquad w x + b = 1$$

Mass (g):
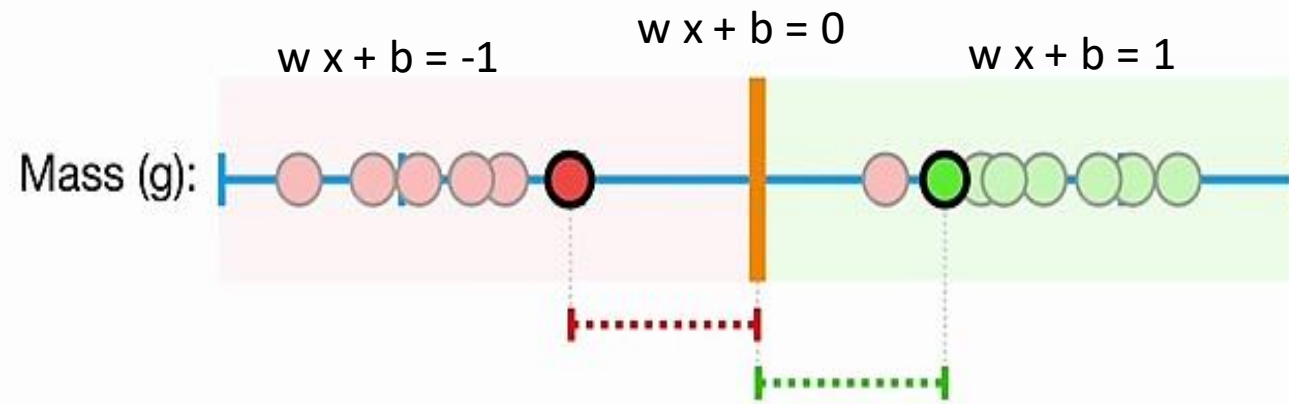
GREEN   if w x + b >= 1
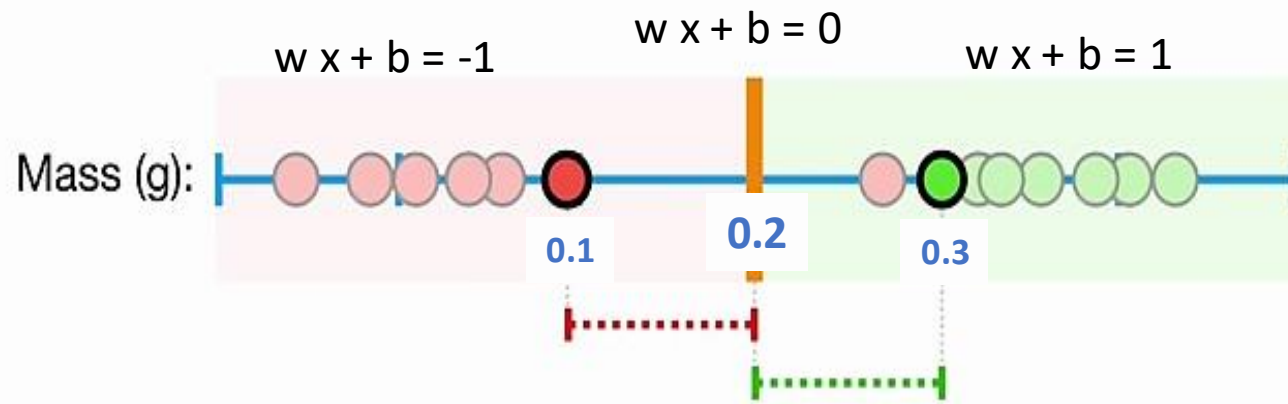
RED   if w x + b <= -1

From One to Two Dimensions

Maximum Margin Hyperplanes

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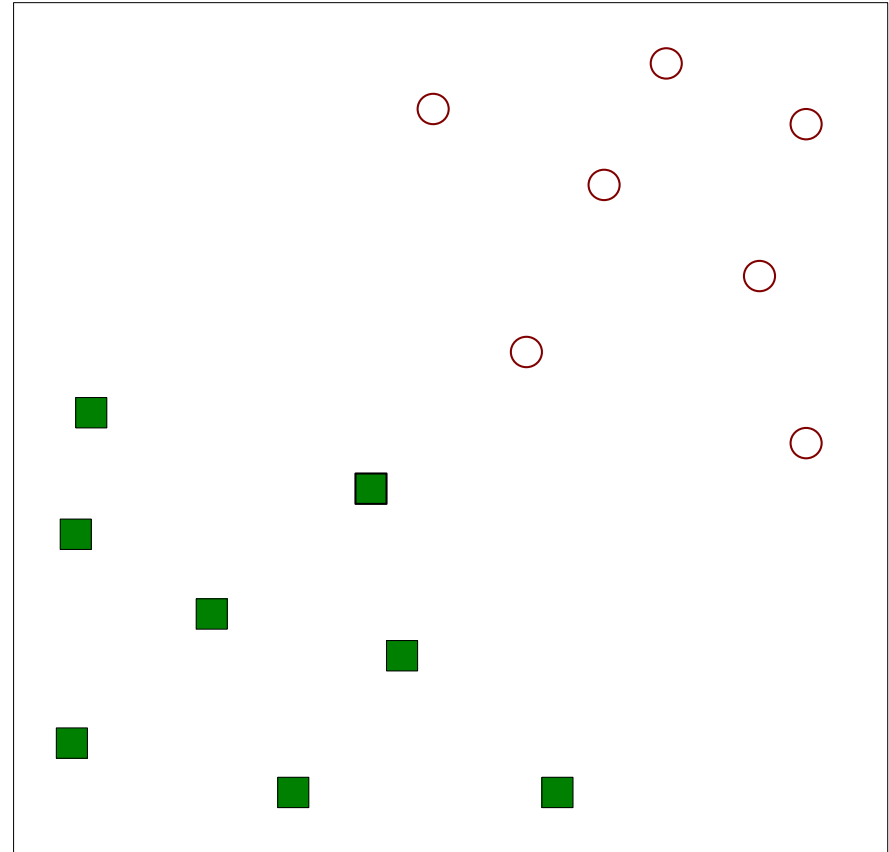
Maximum Margin Hyperplanes

- One possible solution.
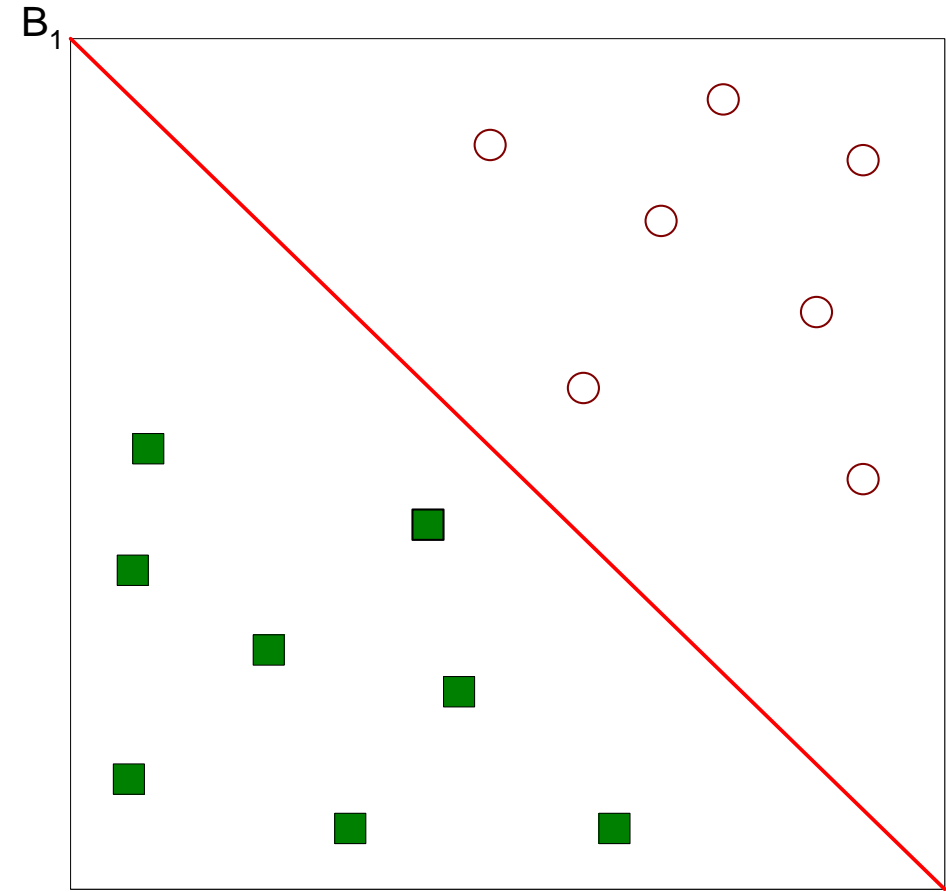
Maximum Margin Hyperplanes

- Another possible solution.
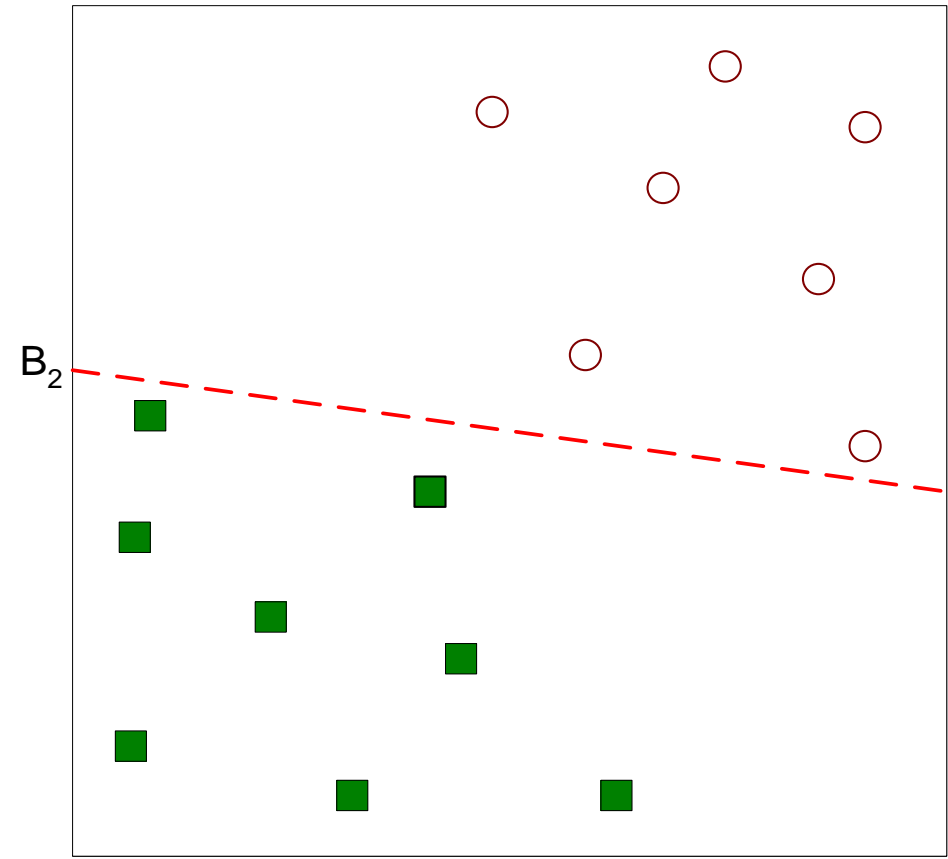
Maximum Margin Hyperplanes

- Other possible solutions.
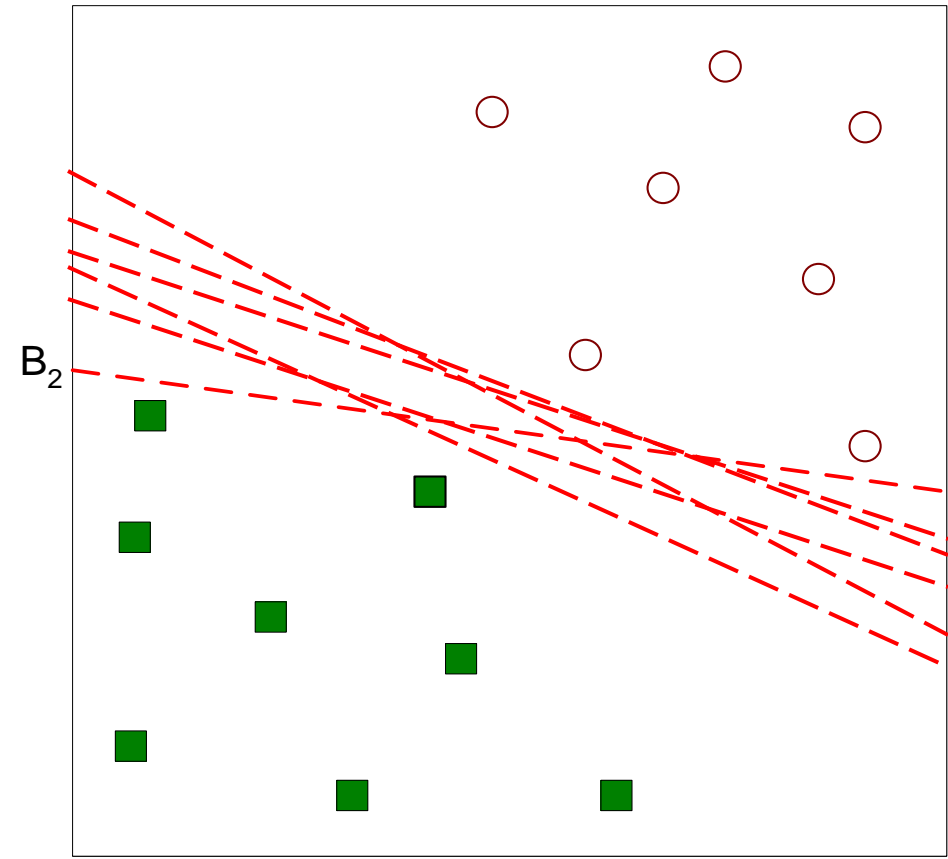
Maximum Margin Hyperplanes

- Let's focus on B_1 and B_2.
- Which one is better?
- How do you define better?
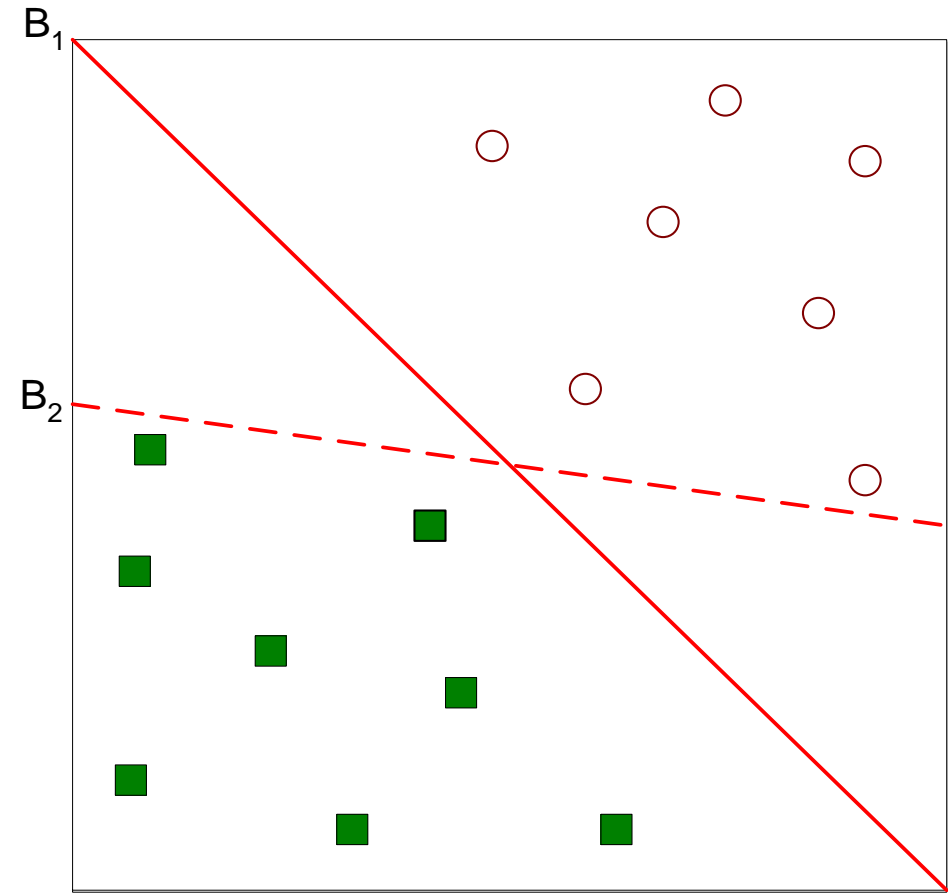
Maximum Margin Hyperplanes

- The best solution is the hyperplane that **maximizes** the **margin**.

- Thus, B_1 is better than B_2.
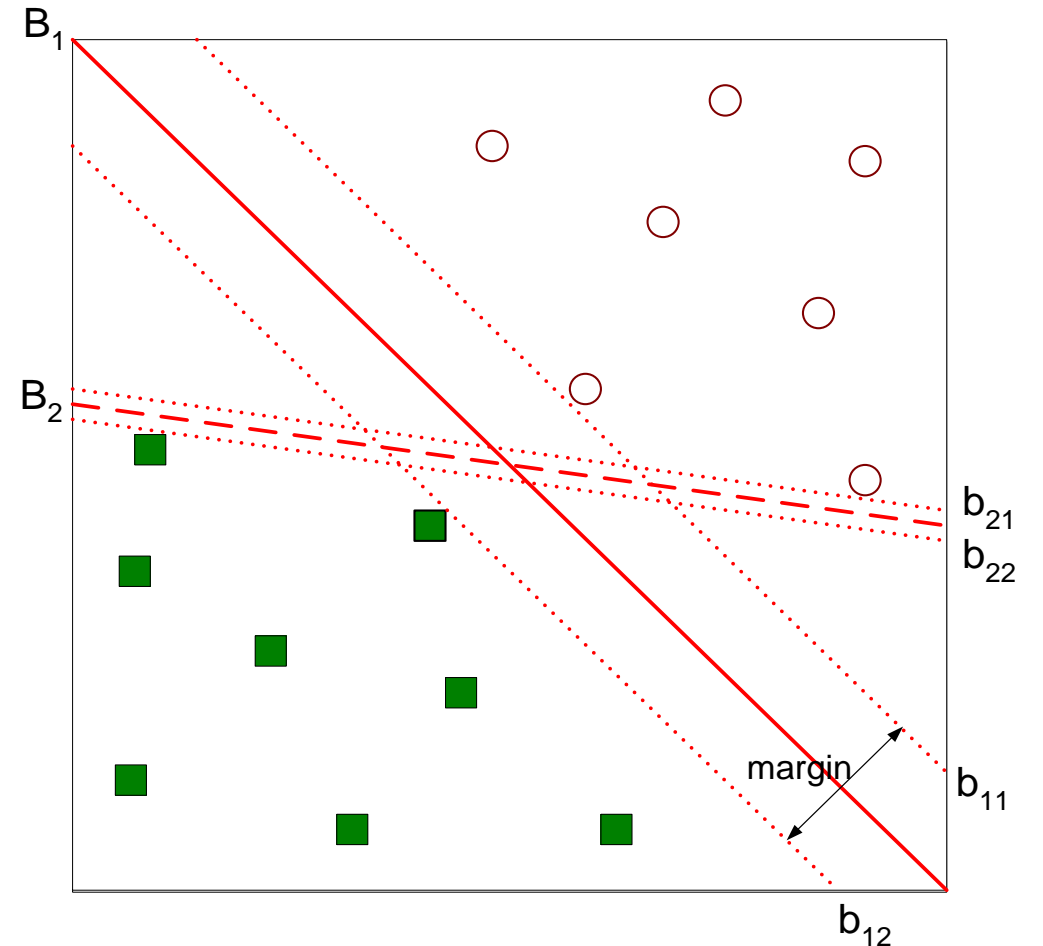
Linear SVM: Separable Case

$$\vec{w} \bullet \vec{x} + b = +1$$

- 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.

decision boundary
$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

support vectors

B_1

b_{11}

b_{12}

- 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
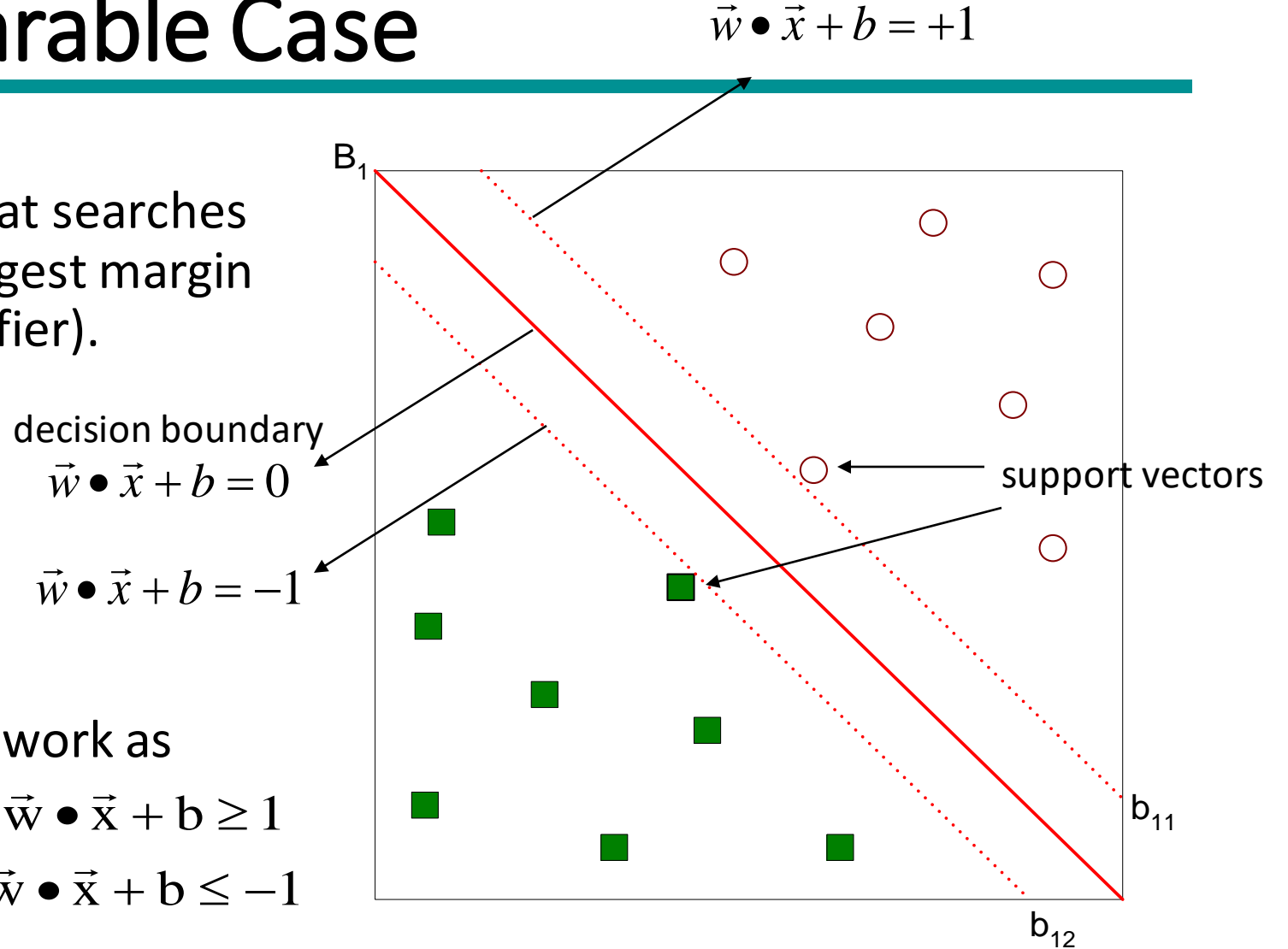
Linear SVM: Separable Case

$$\vec{w} \bullet \vec{x} + b = +1$$

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

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

decision boundary
$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

B₁

support vectors

b₁₁

b₁₂

Linear SVM: Separable Case

$$\vec{w} \bullet \vec{x} + b = +1$$
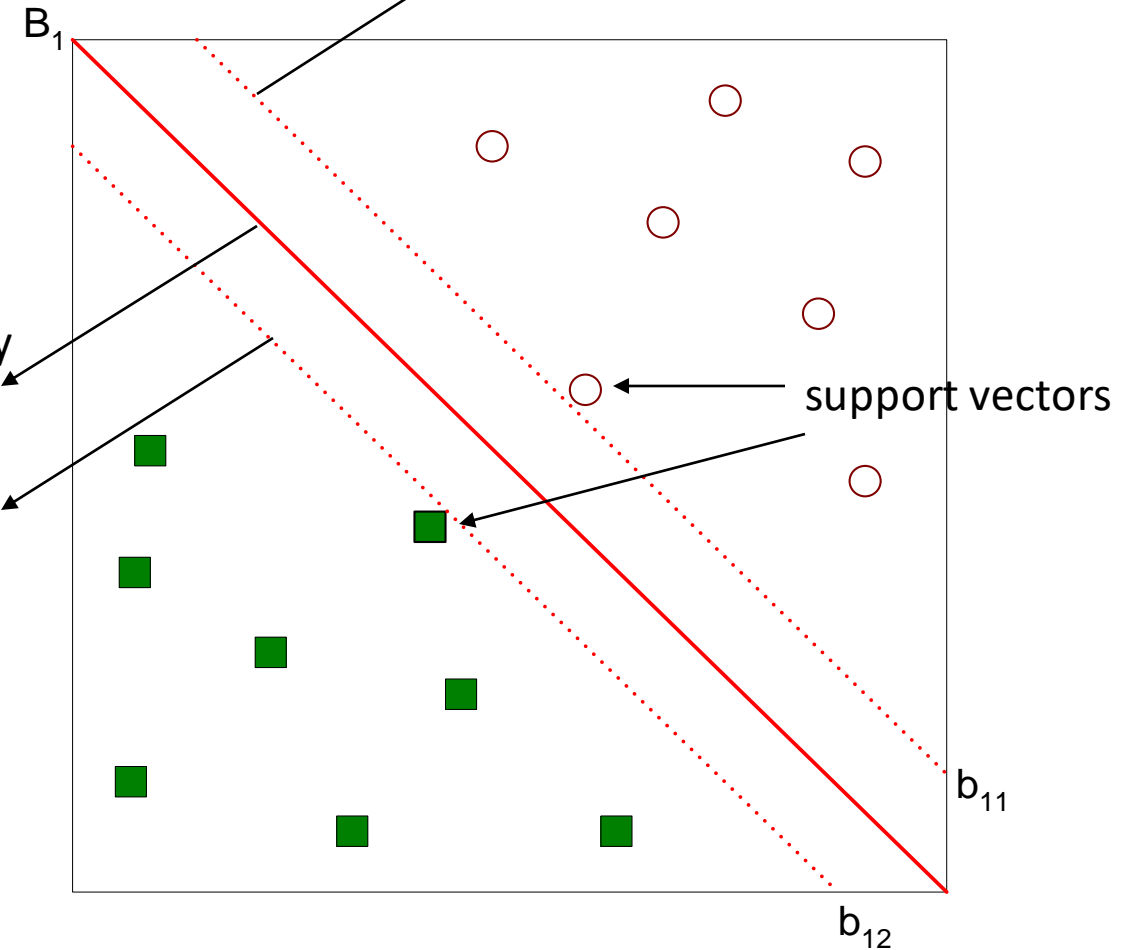
- The distance between B_1 and b_{11} is $1/\|w\|$

- 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 $\|w\|$

B_1

decision boundary
$$\vec{w} \bullet \vec{x} + b = 0$$

$$\vec{w} \bullet \vec{x} + b = -1$$

margin

b_{11}

b_{12}
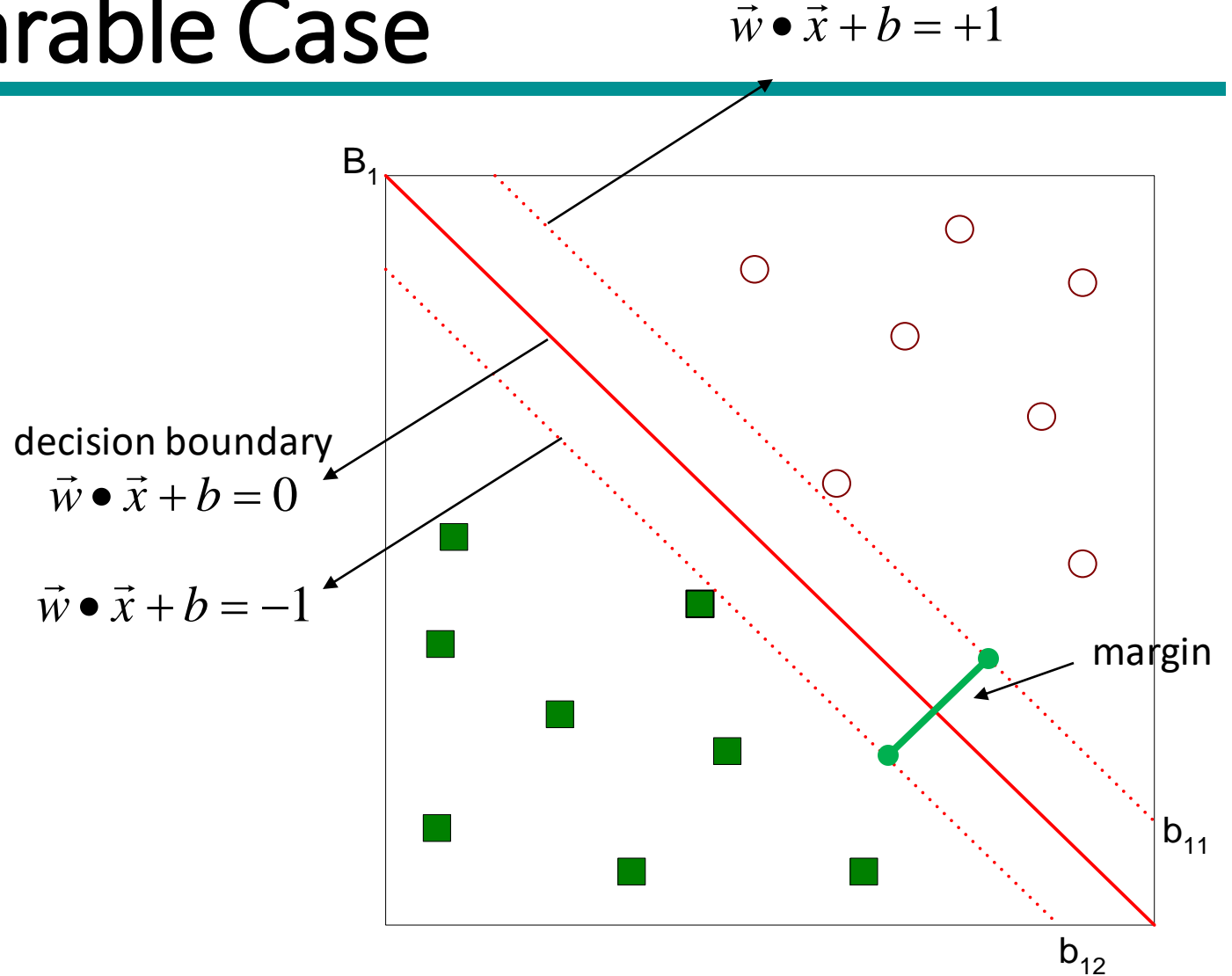
Learning a Linear SVM

- 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)

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

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

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

=

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

Constrained Optimization Problem

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

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

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 :

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

subject to $\sum_i y_i \alpha_i = 0$ 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
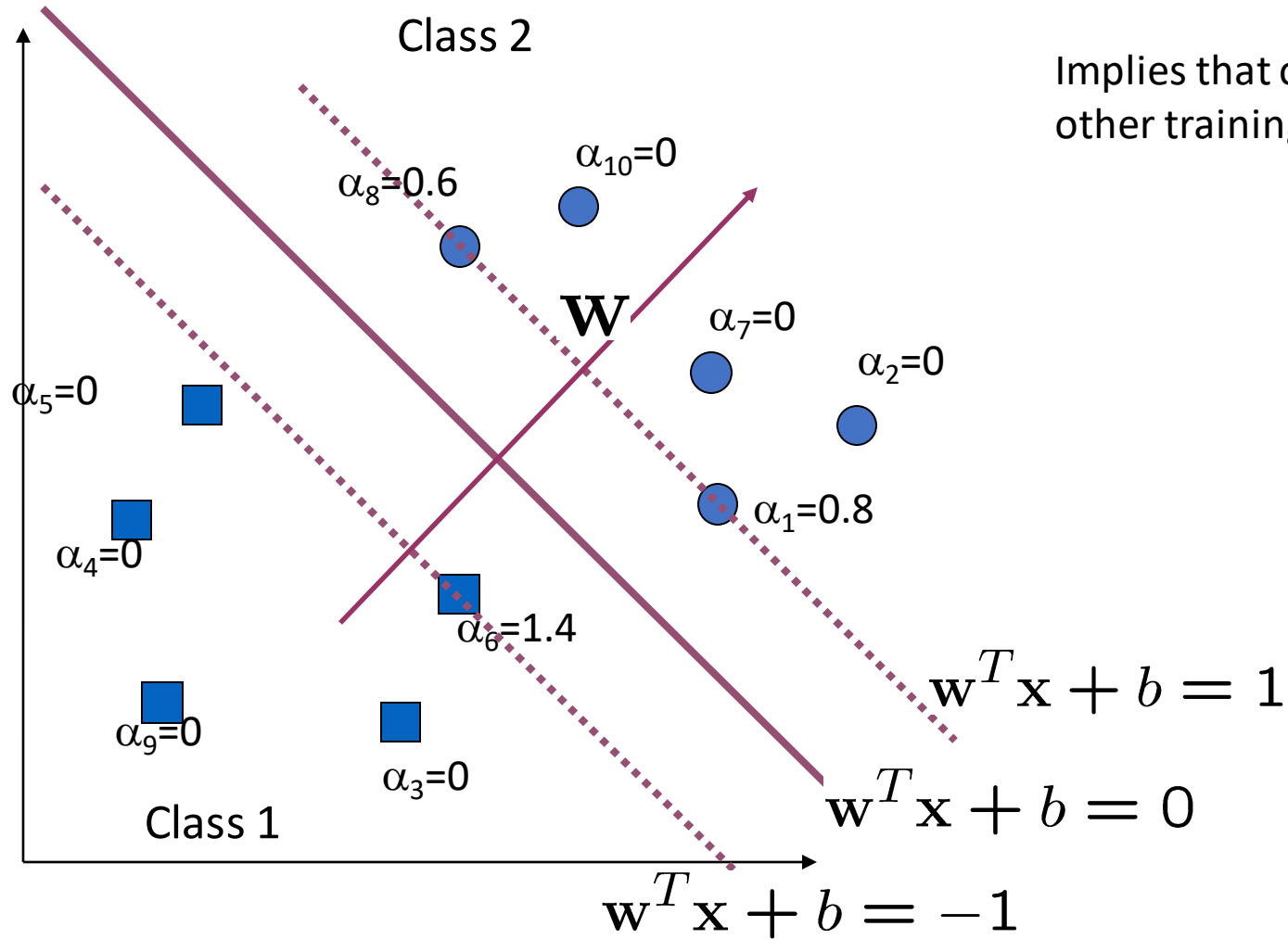
Implies that only support vectors matter; other training examples are ignorable.

Class 2

$\alpha_{10}=0$

$\alpha_8=0.6$

$\mathbf{W}$

$\alpha_7=0$

$\alpha_2=0$

$\alpha_5=0$

$\alpha_1=0.8$

$\alpha_4=0$

$\alpha_6=1.4$

$\alpha_9=0$

$\alpha_3=0$

Class 1

$$\mathbf{w}^T\mathbf{x} + b = 1$$

$$\mathbf{w}^T\mathbf{x} + b = 0$$

$$\mathbf{w}^T\mathbf{x} + b = -1$$

Example of Linear SVM
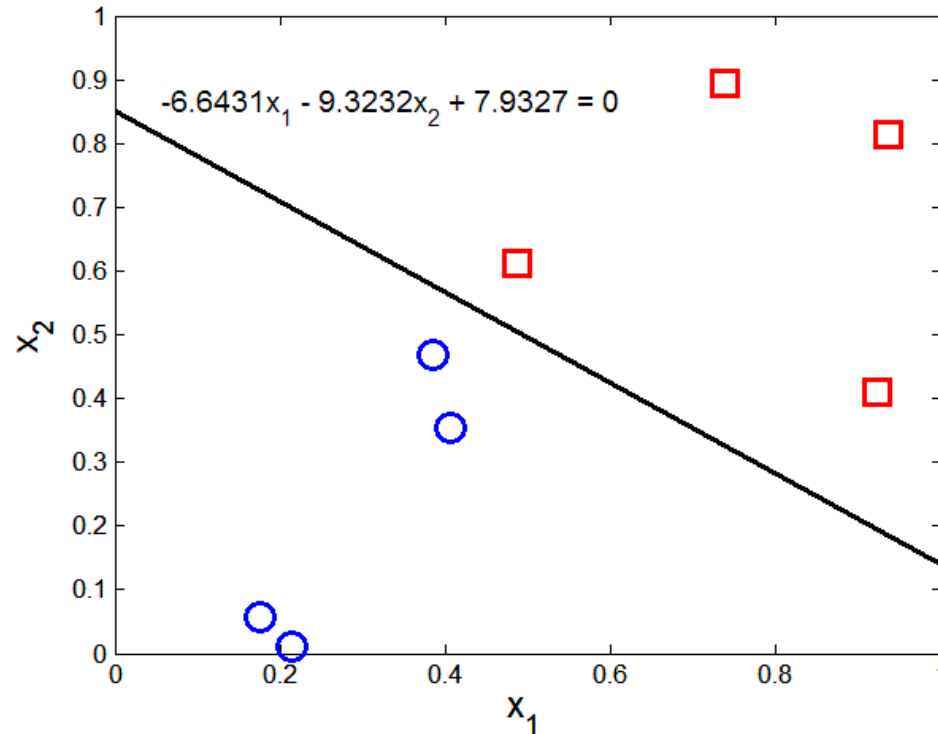
Support vectors

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

$\lambda = \alpha$

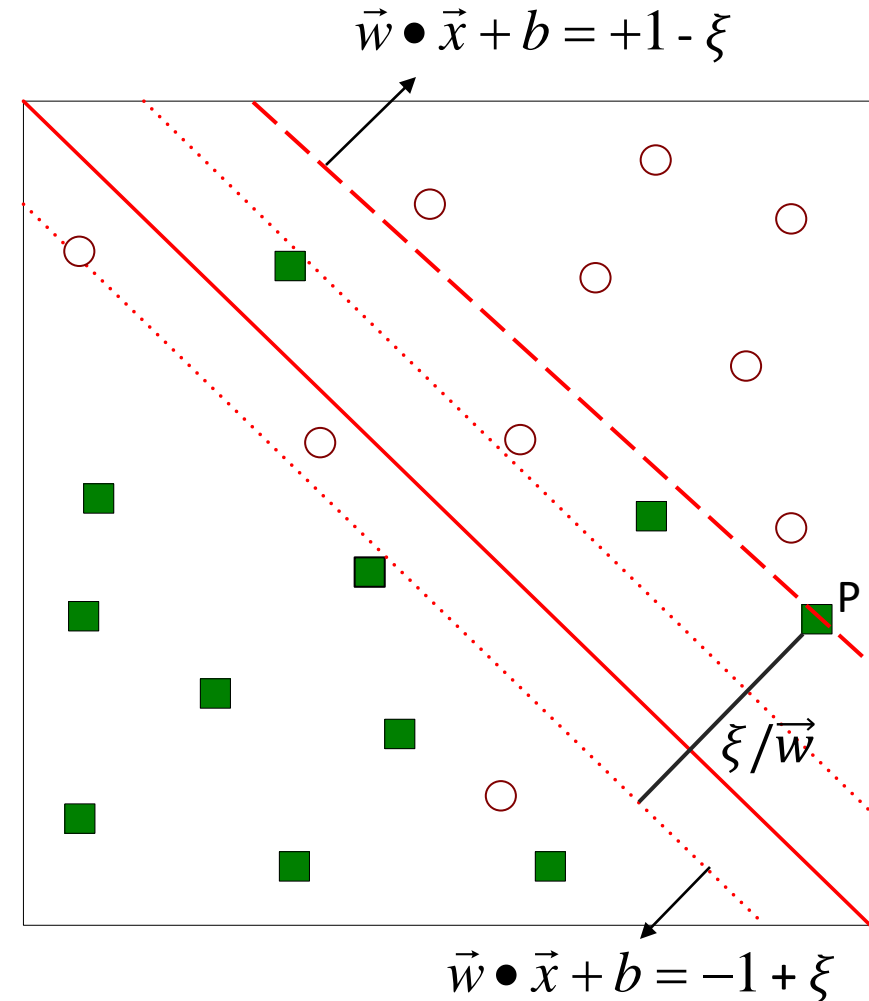# Linear SVM: Non-separable Case

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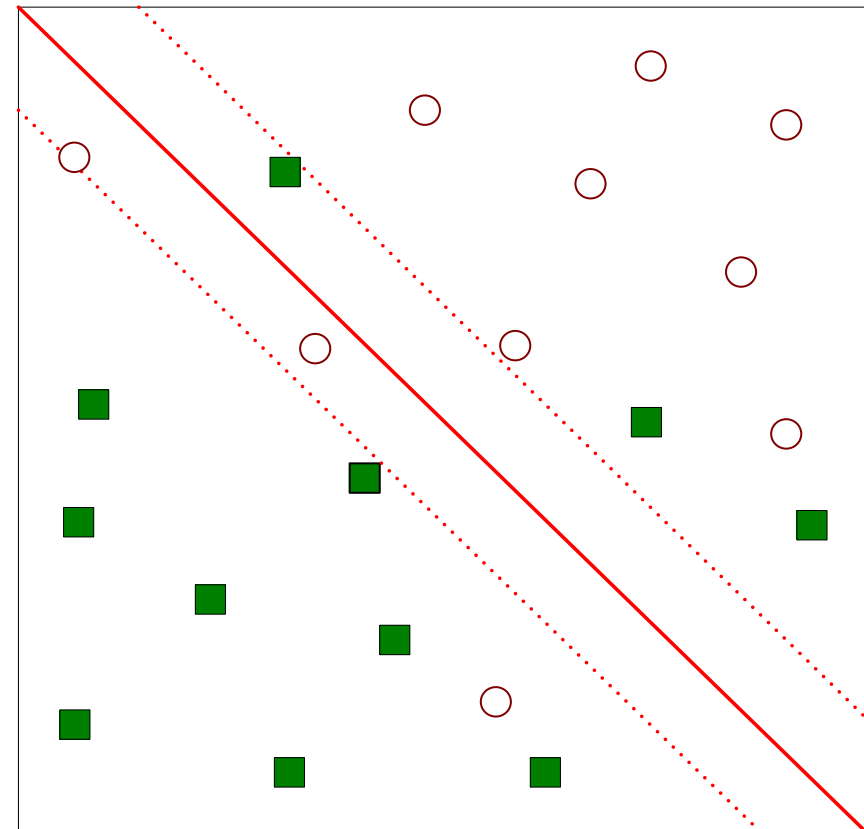
Slack Variables

- 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.

$$\vec{w} \bullet \vec{x} + b = +1 - \xi$$

$$\xi/\vec{w}$$

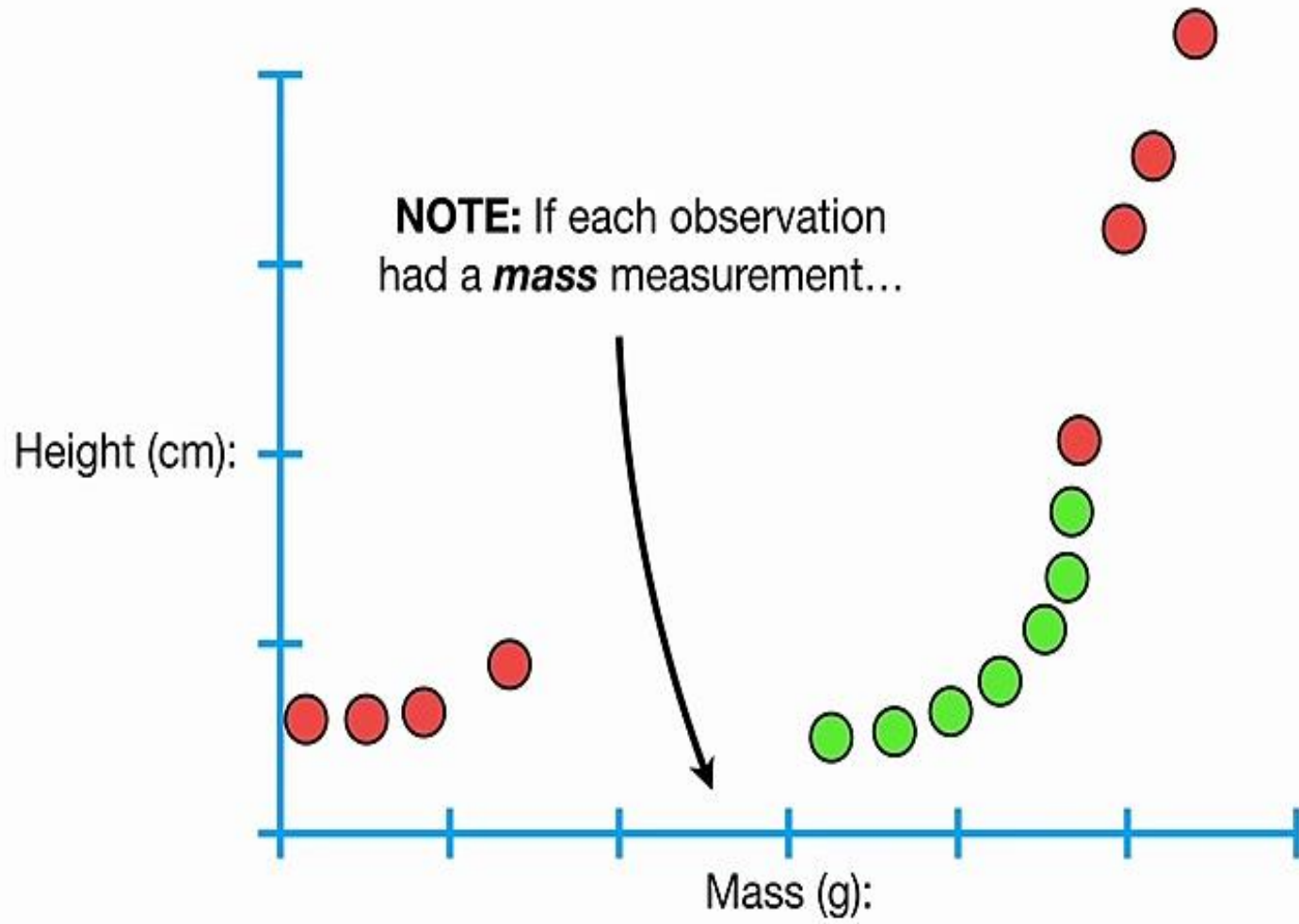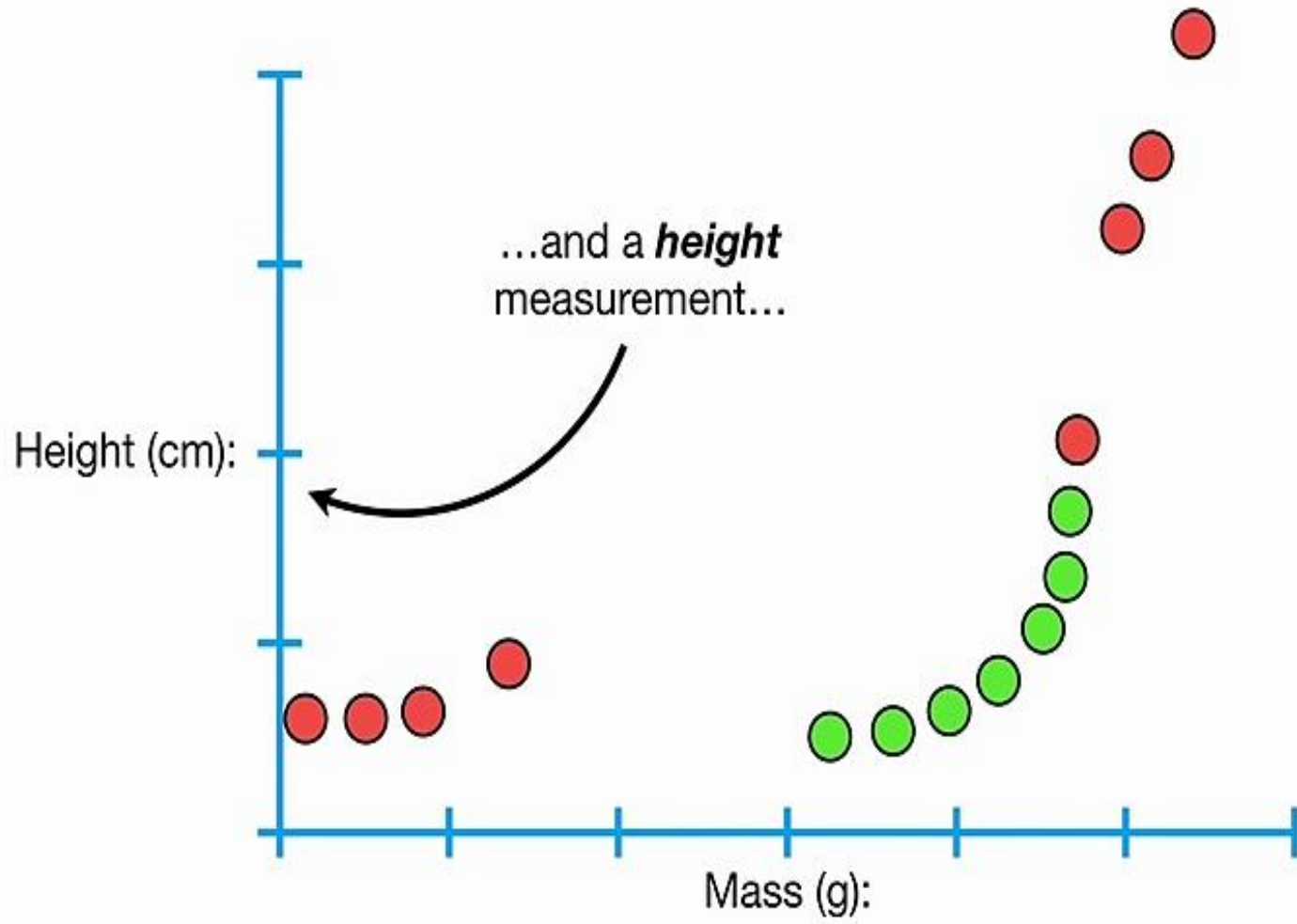$$\vec{w} \bullet \vec{x} + b = -1 + \xi$$
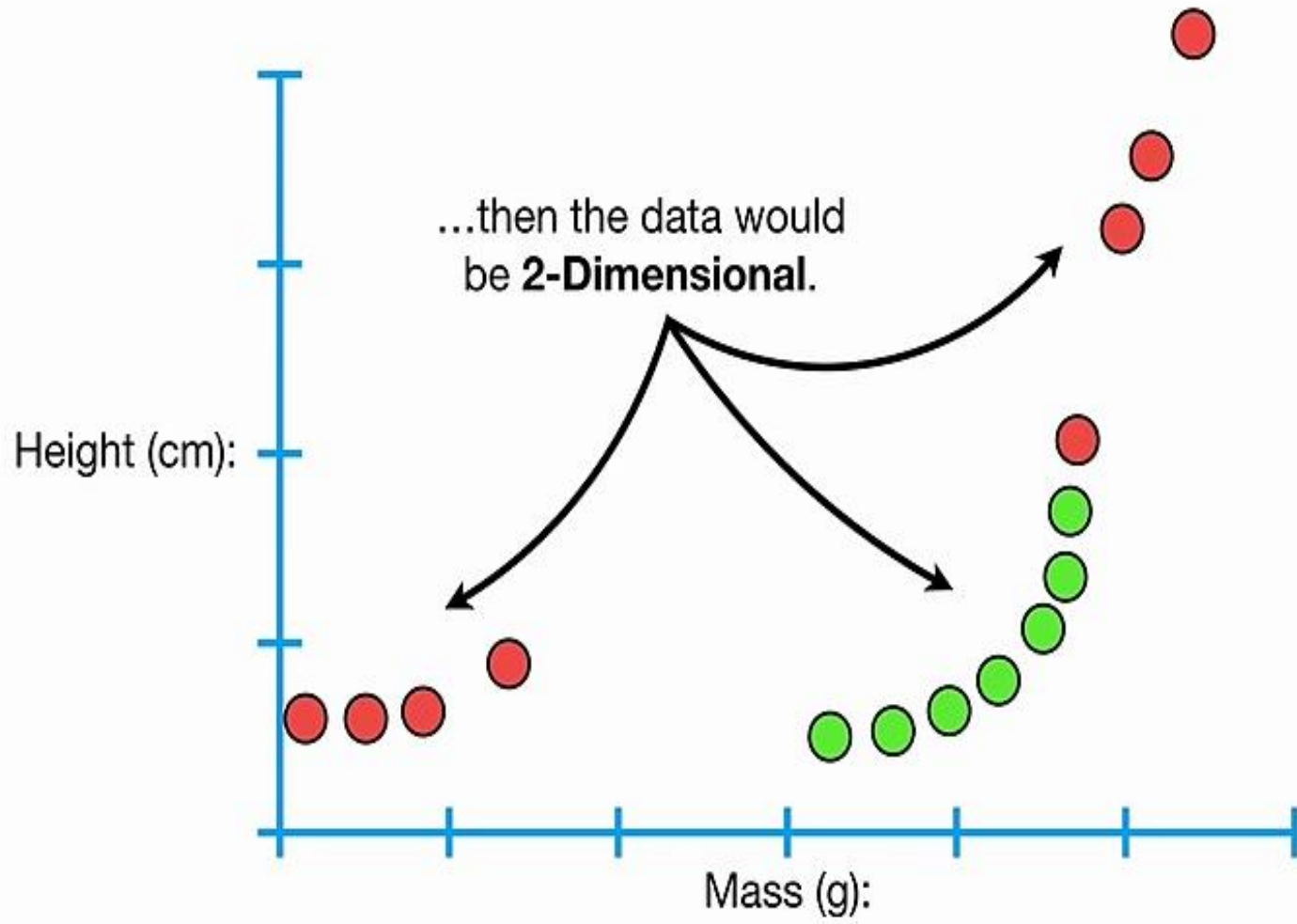
P

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 $0 \leq \lambda \leq C$.

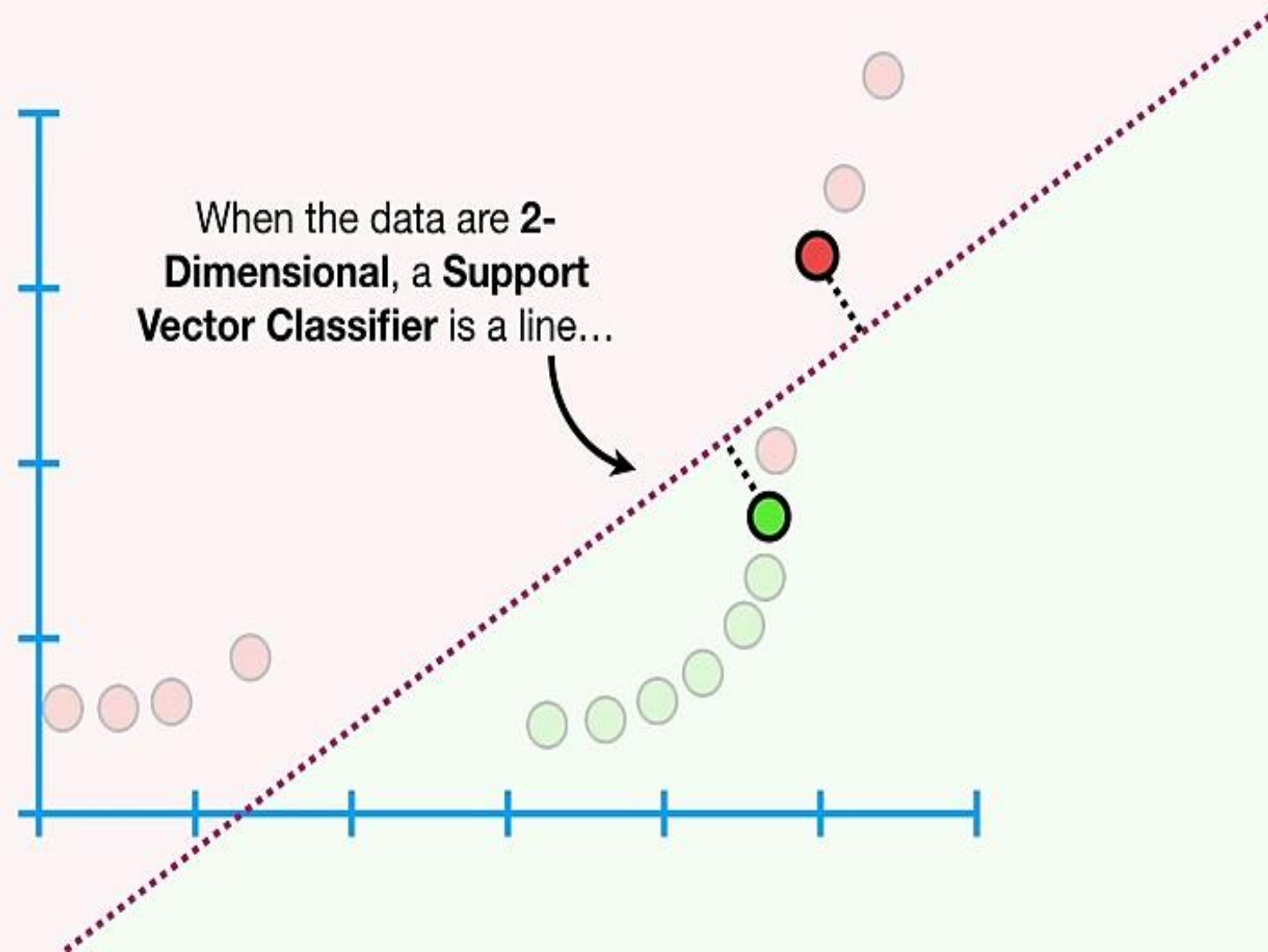$$L(w) = \frac{\|\vec{w}\|^2}{2} + C\left(\sum_{i=1}^{N} \xi_i^k\right)$$

$$y_i = \begin{cases} 1 & \text{if } \vec{w} \bullet \vec{x}_i + b \geq 1 - \xi_i \\ -1 & \text{if } \vec{w} \bullet \vec{x}_i + b \leq -1 + \xi_i \end{cases}$$
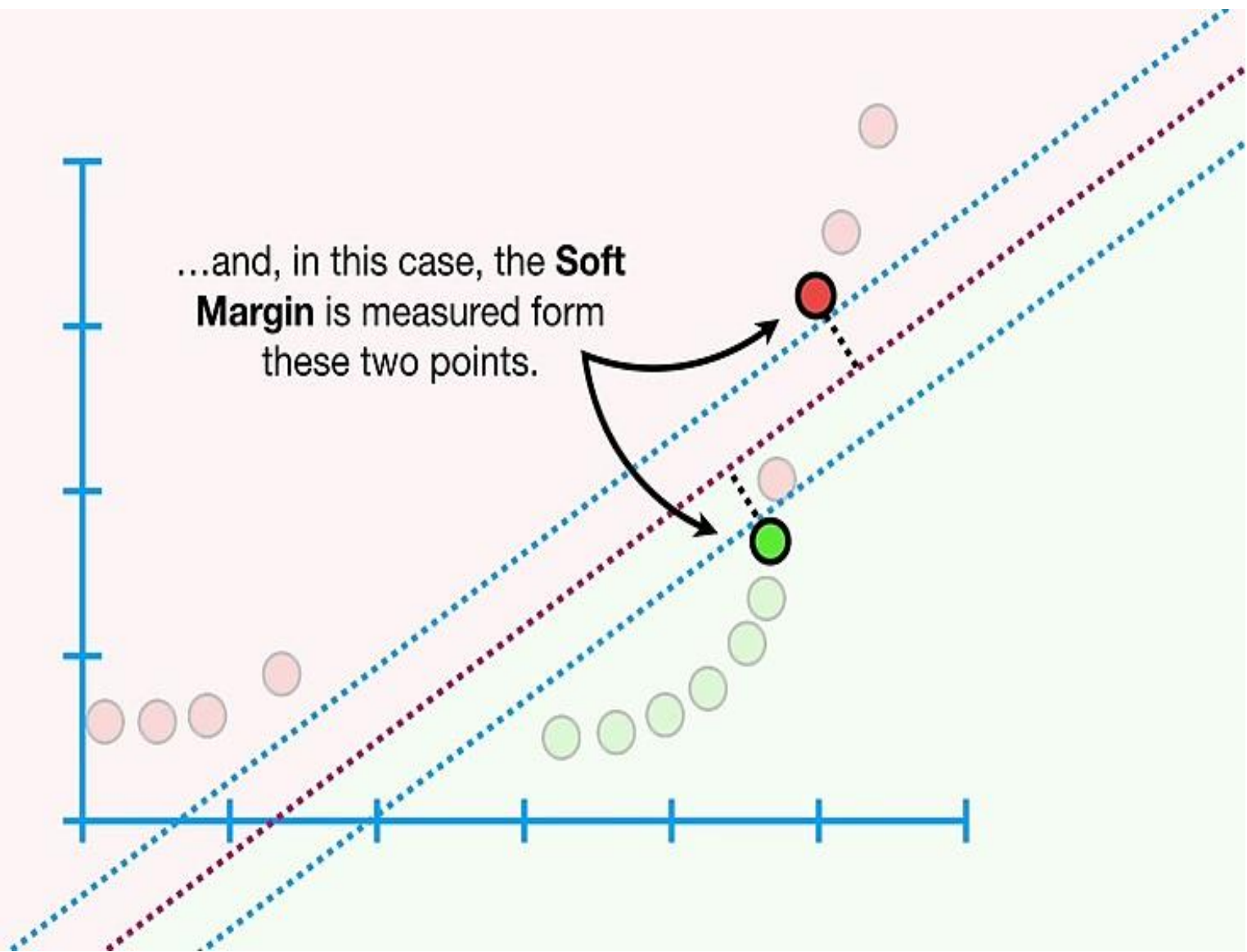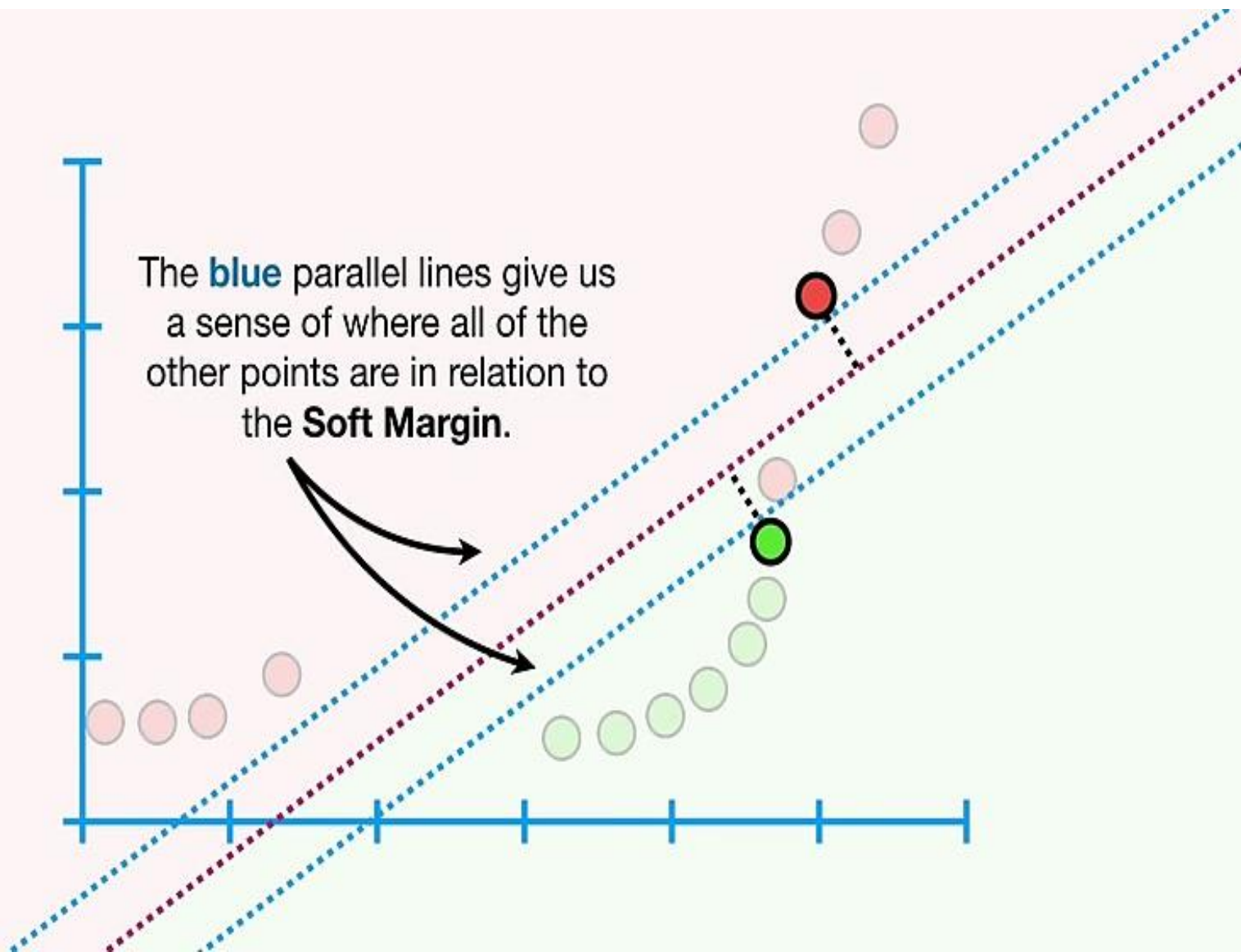
When the data are **2-Dimensional**, a **Support Vector Classifier** is a line...

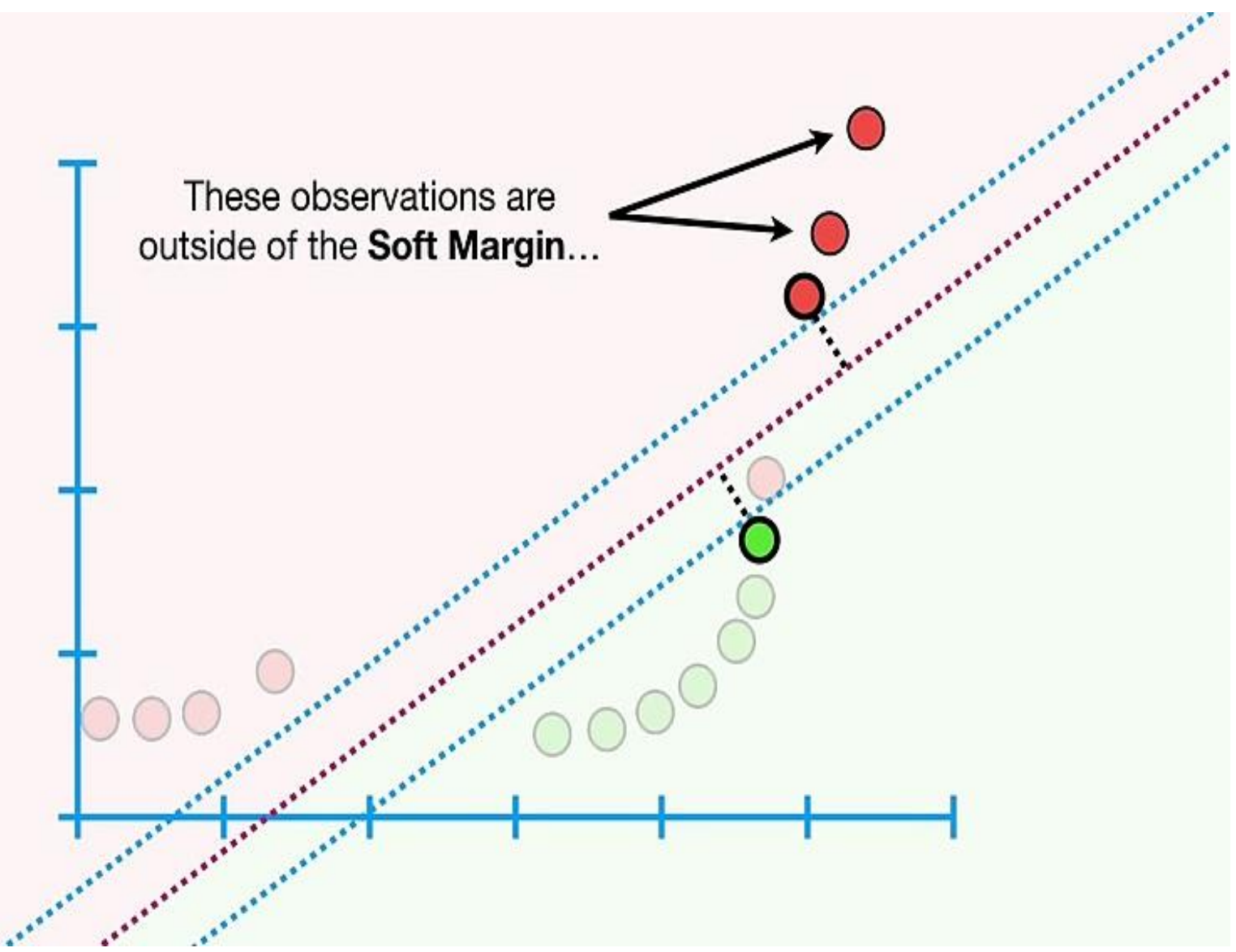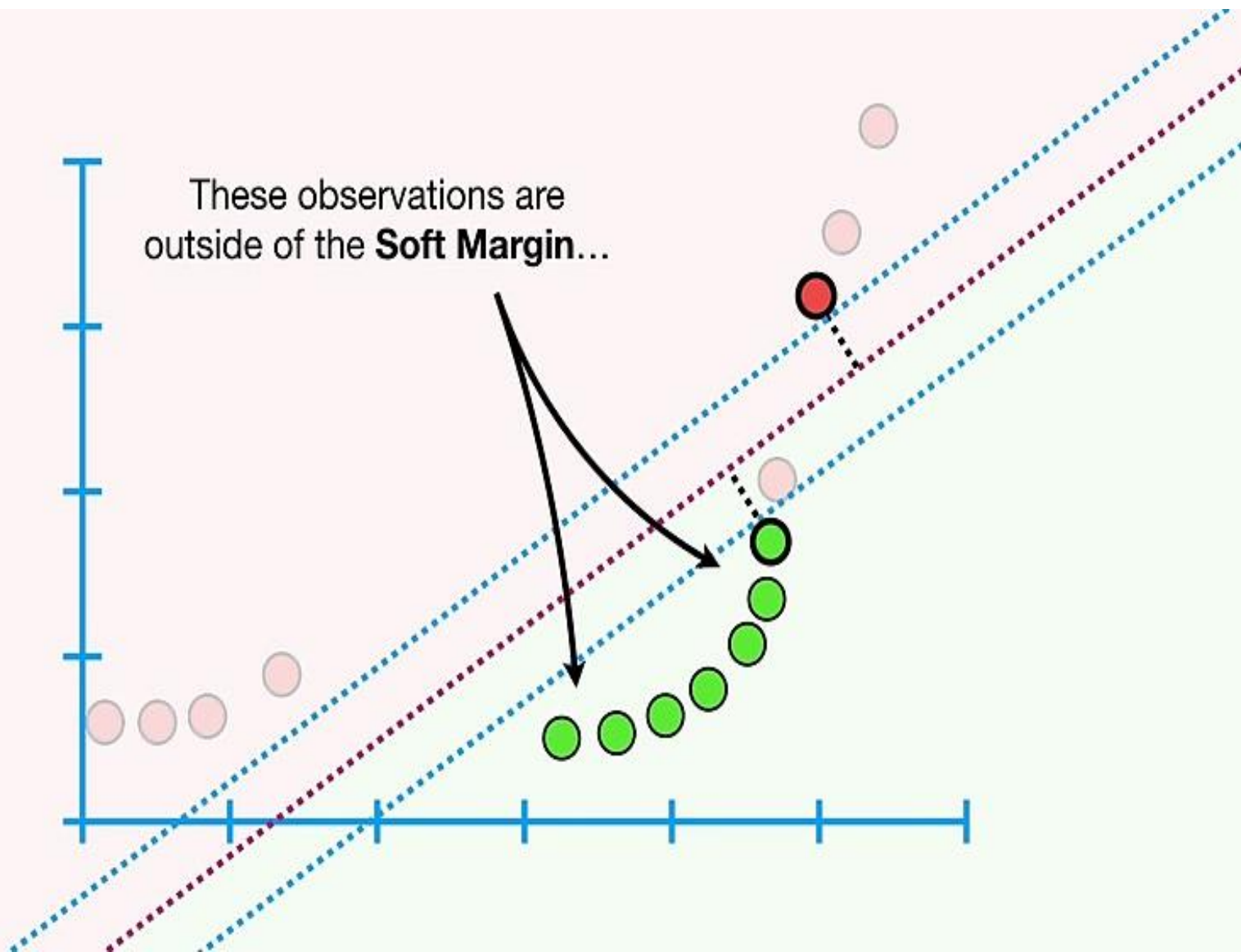...and, in this case, the **Soft Margin** is measured form these two points.

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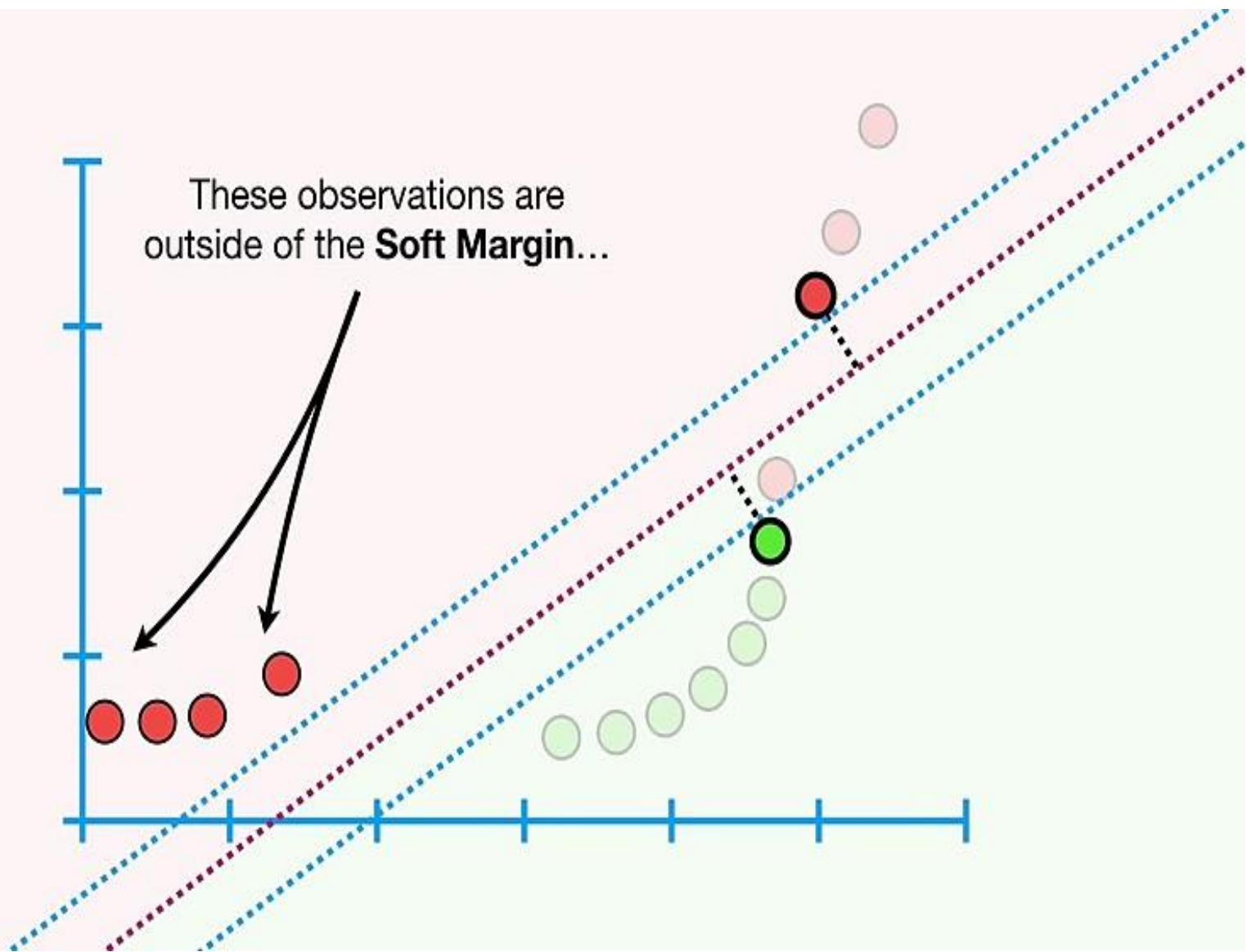
These observations are outside of the **Soft Margin**…
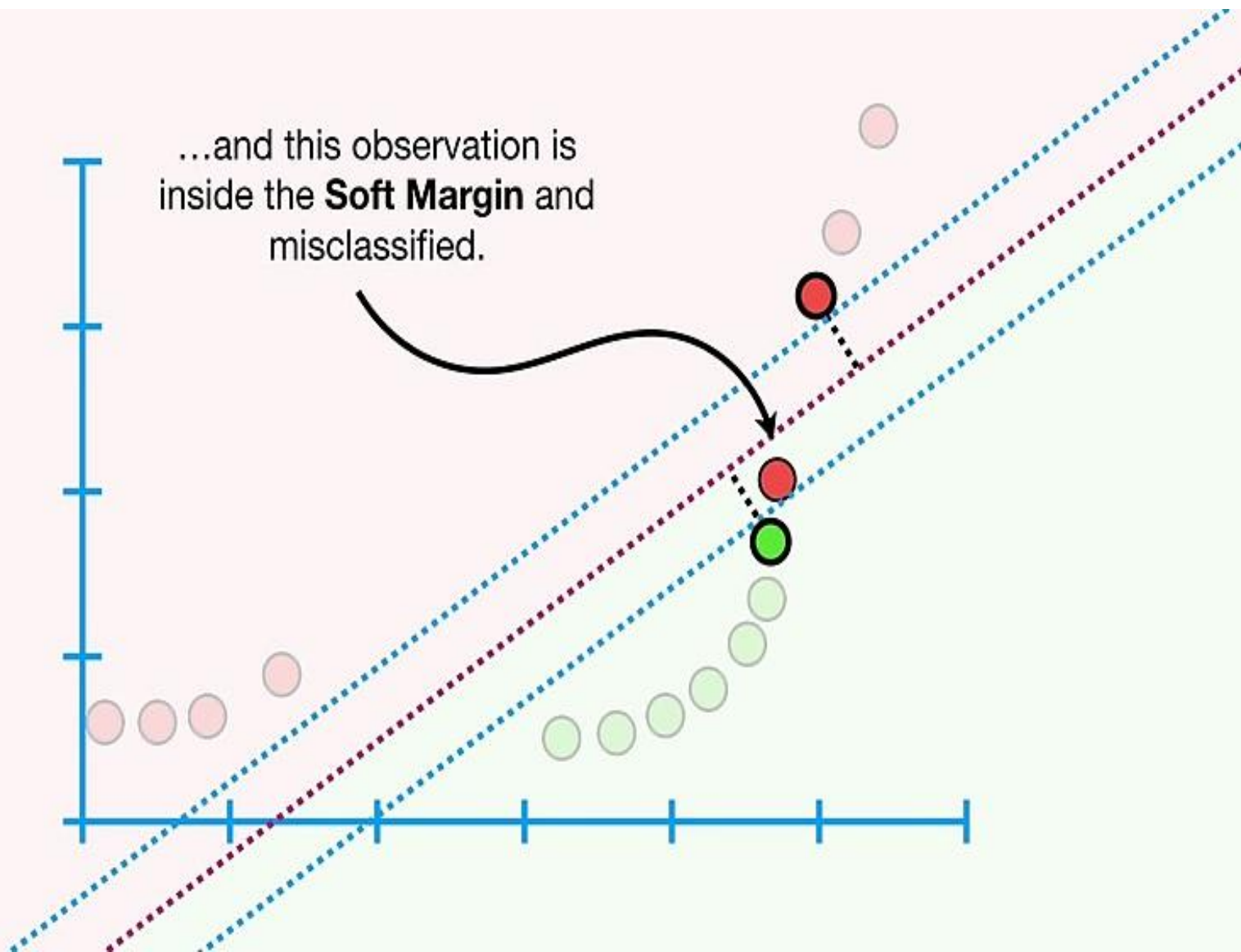
These observations are outside of the **Soft Margin**...
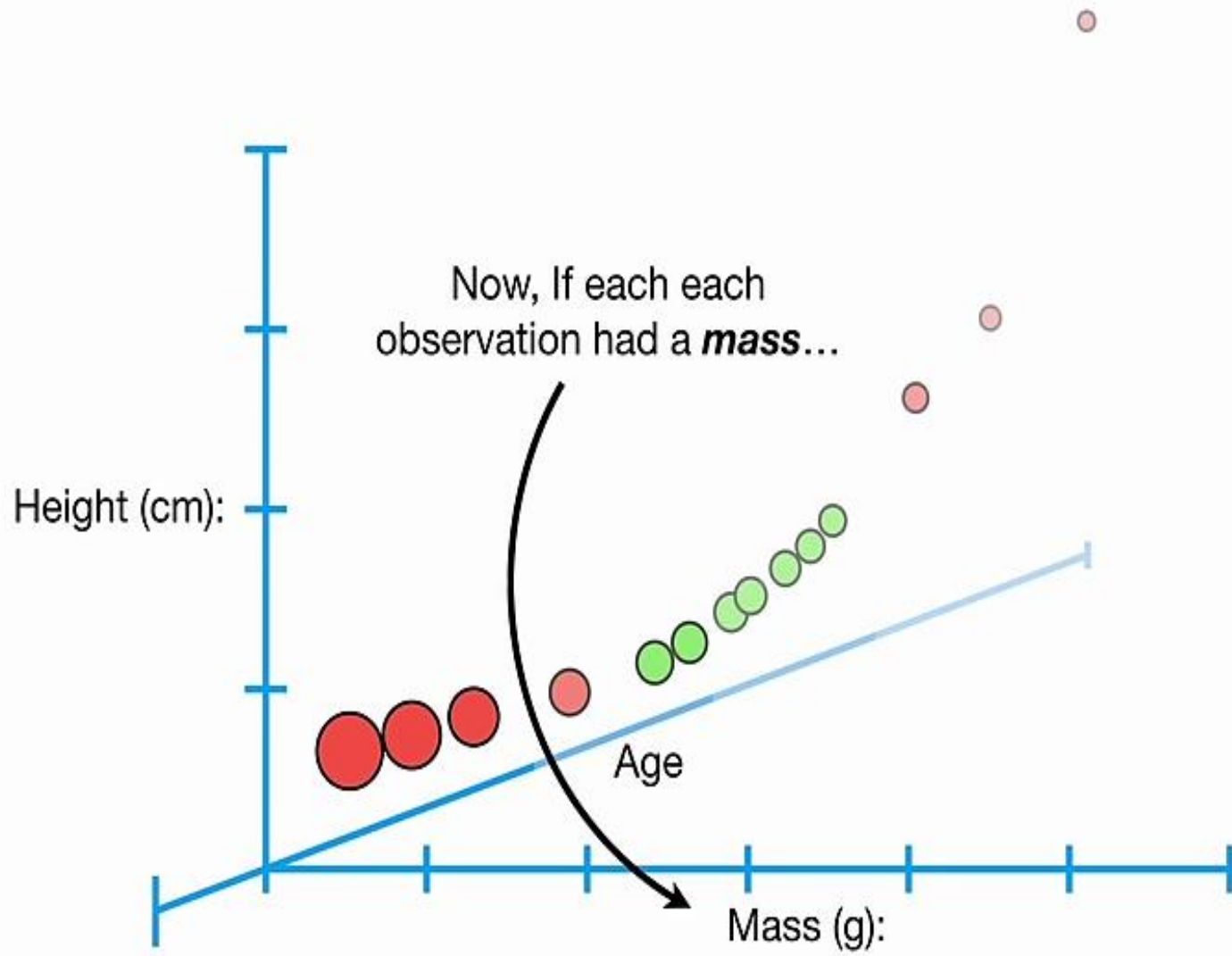
These observations are outside of the **Soft Margin**…

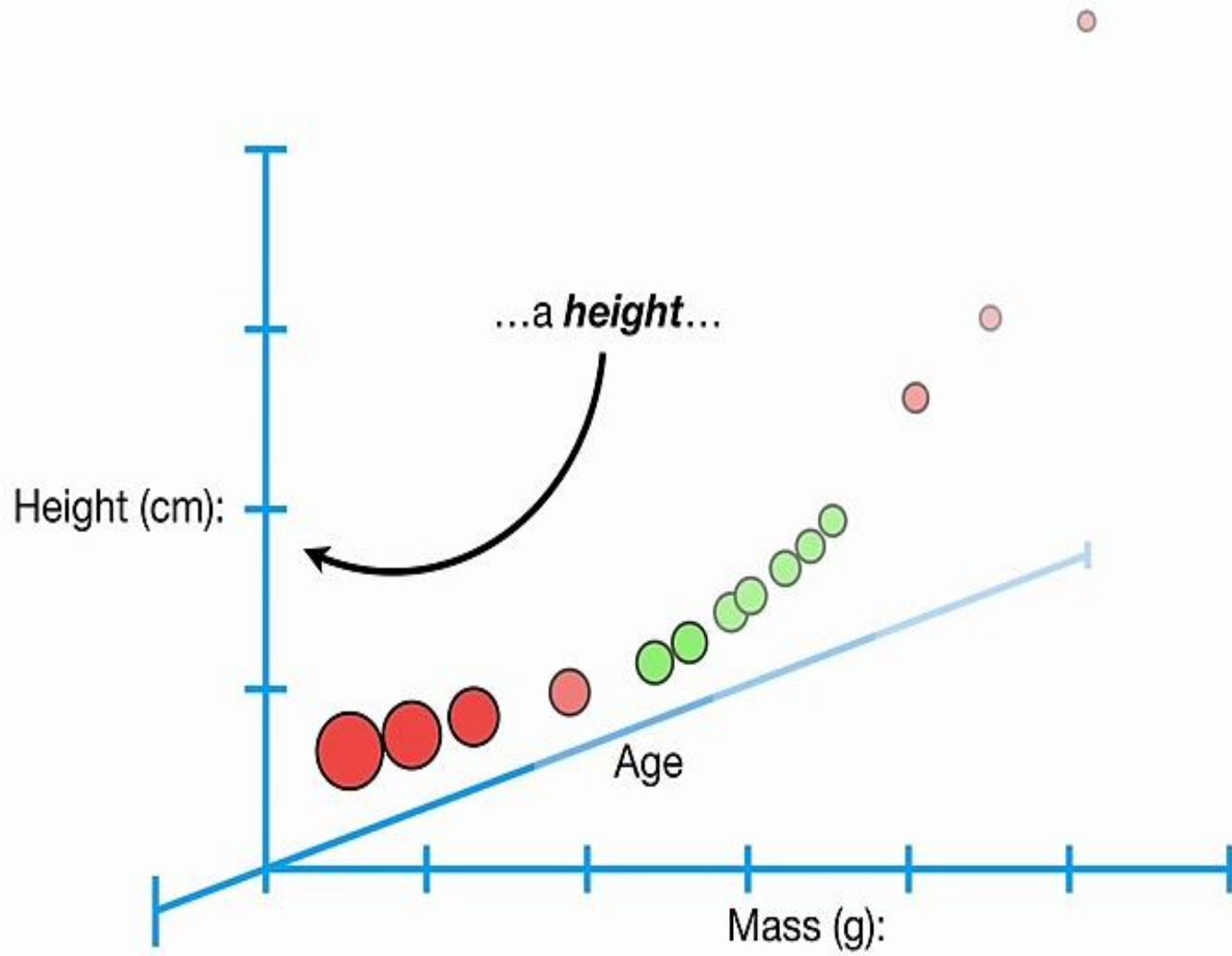...and this observation is inside the **Soft Margin** and misclassified.
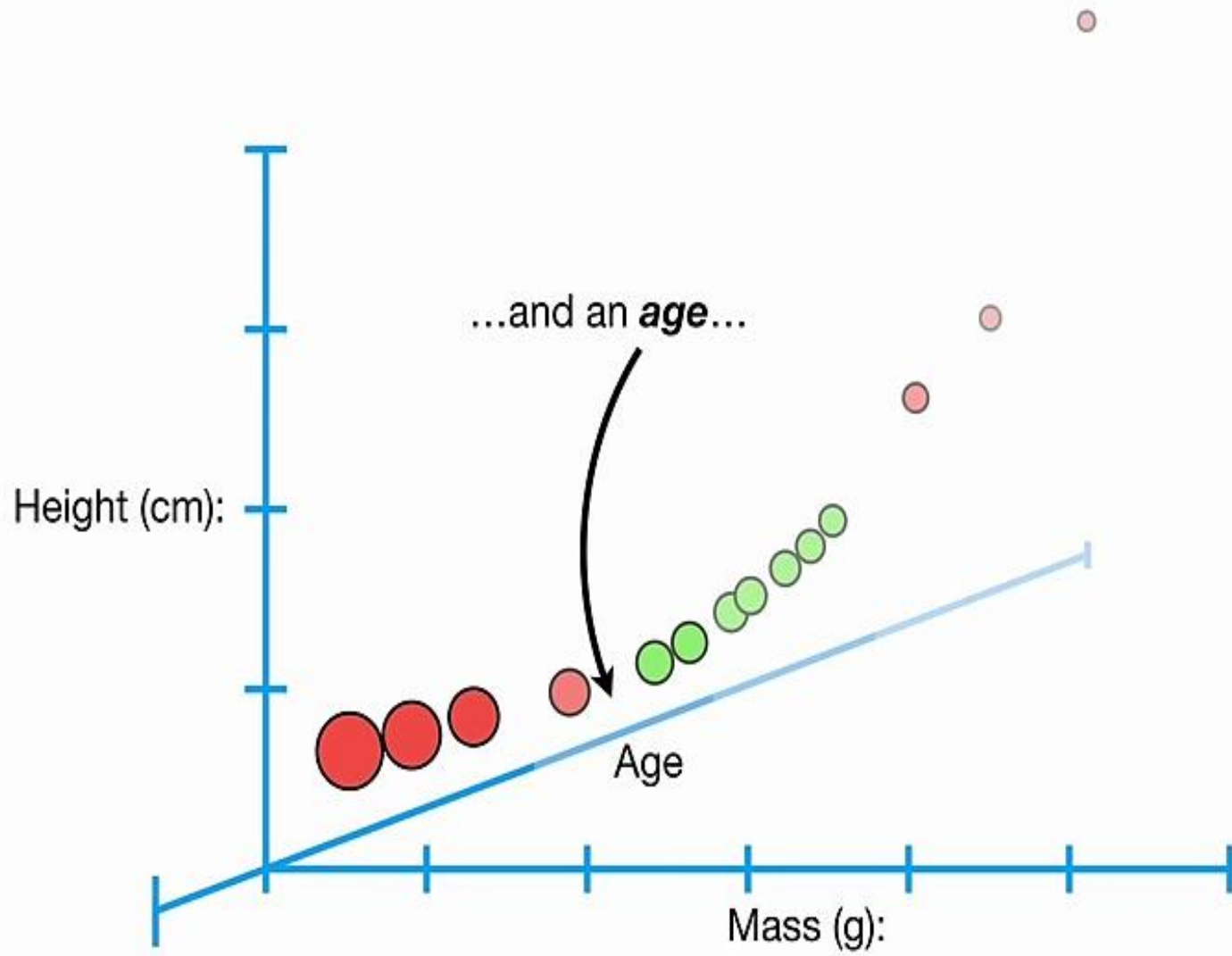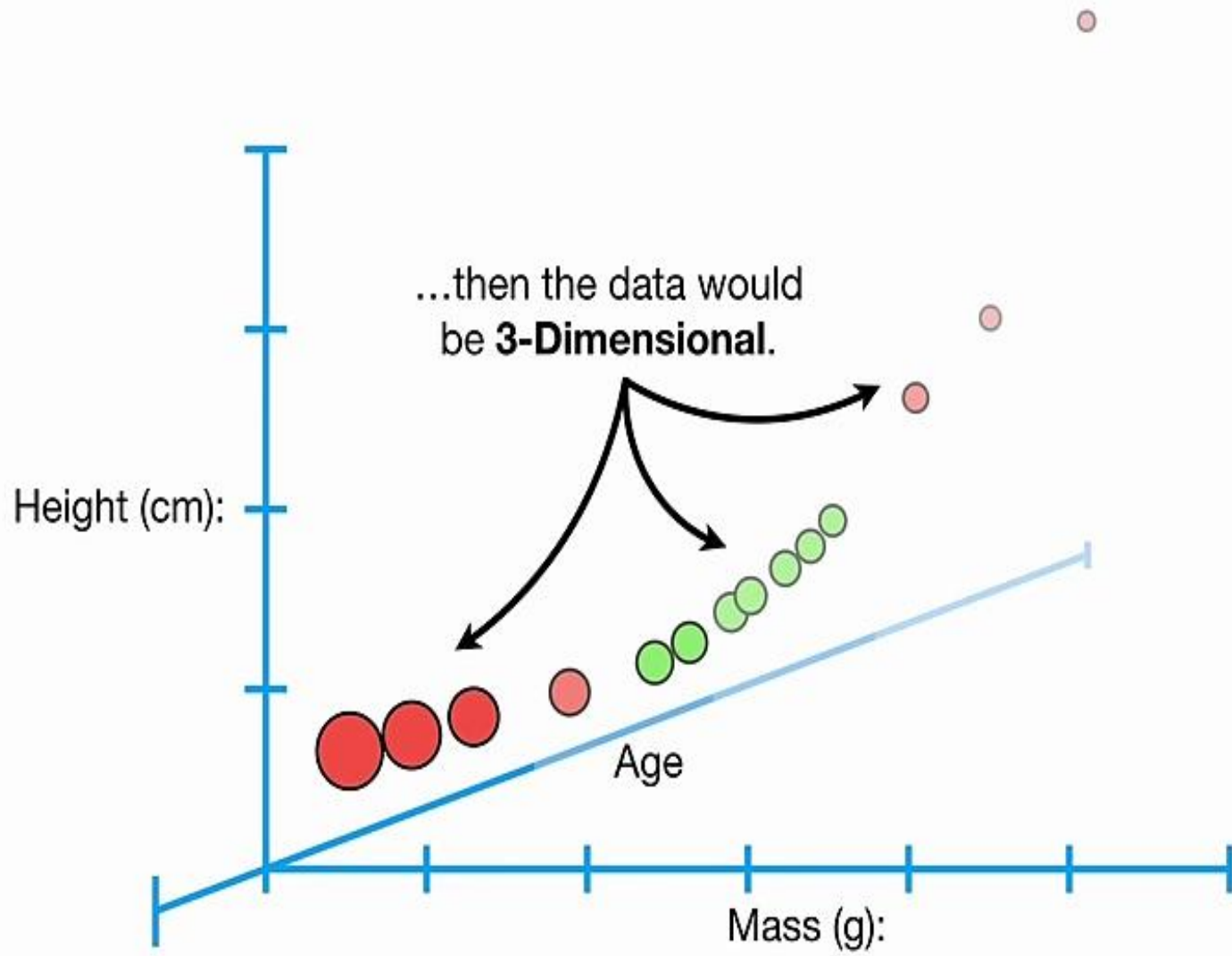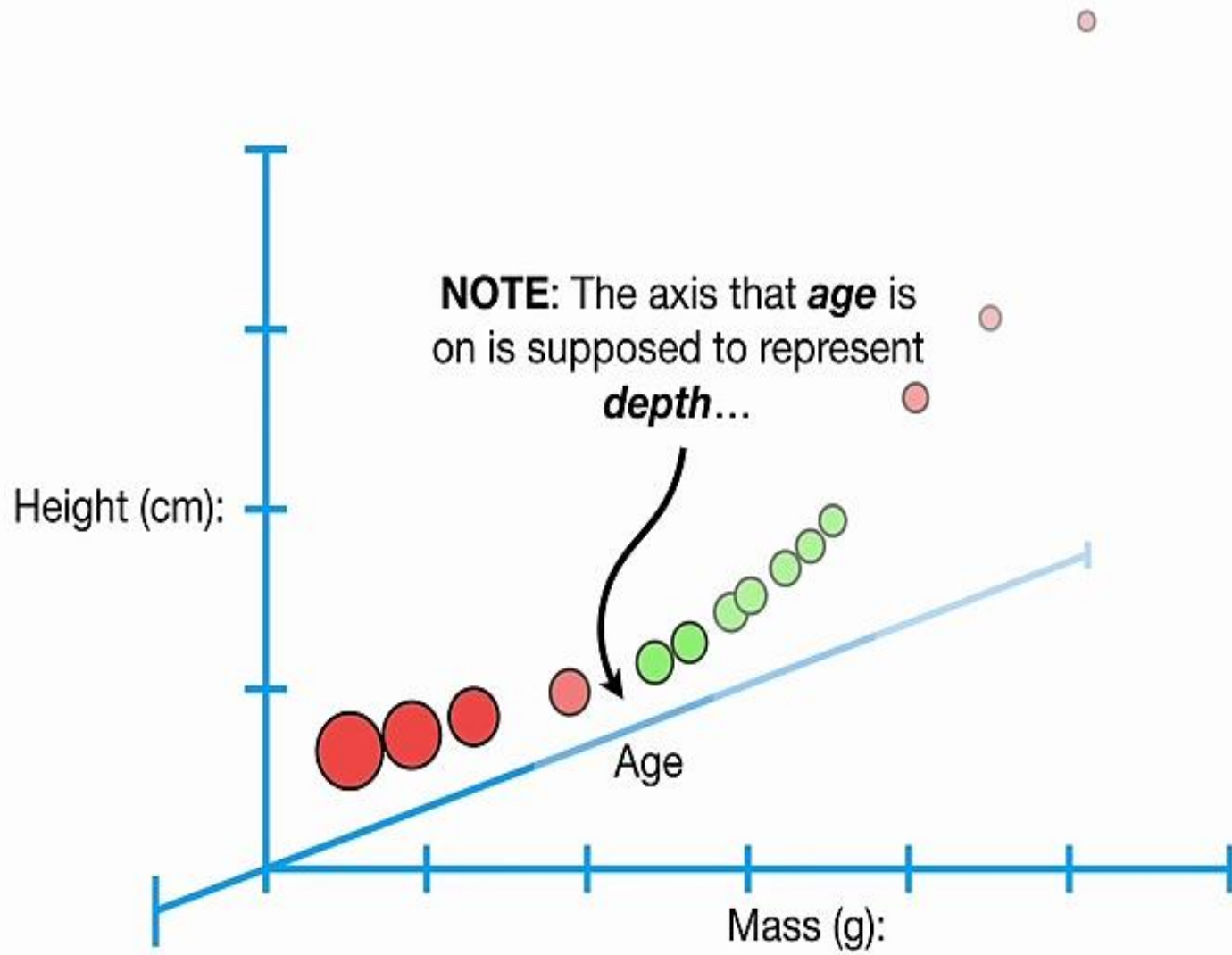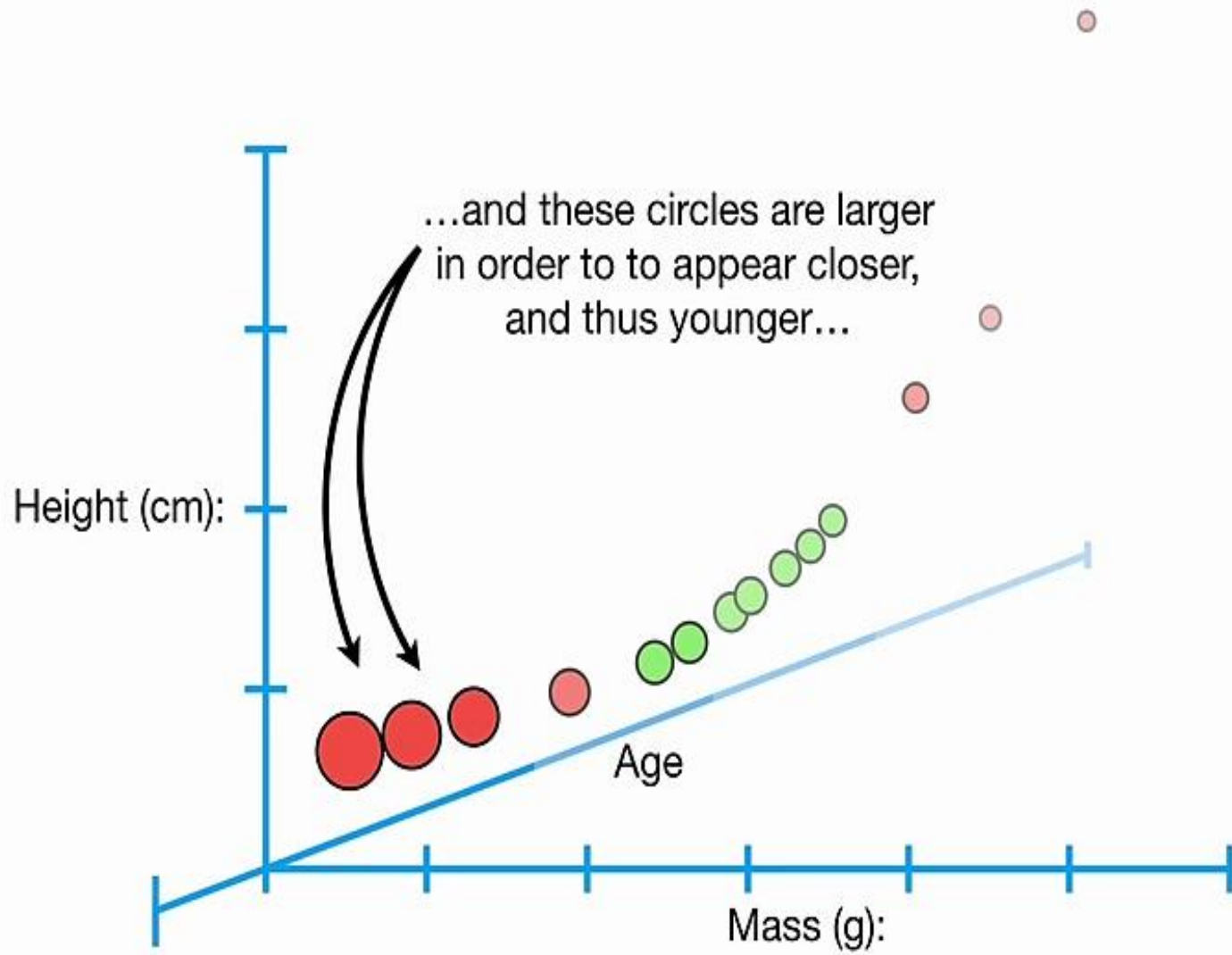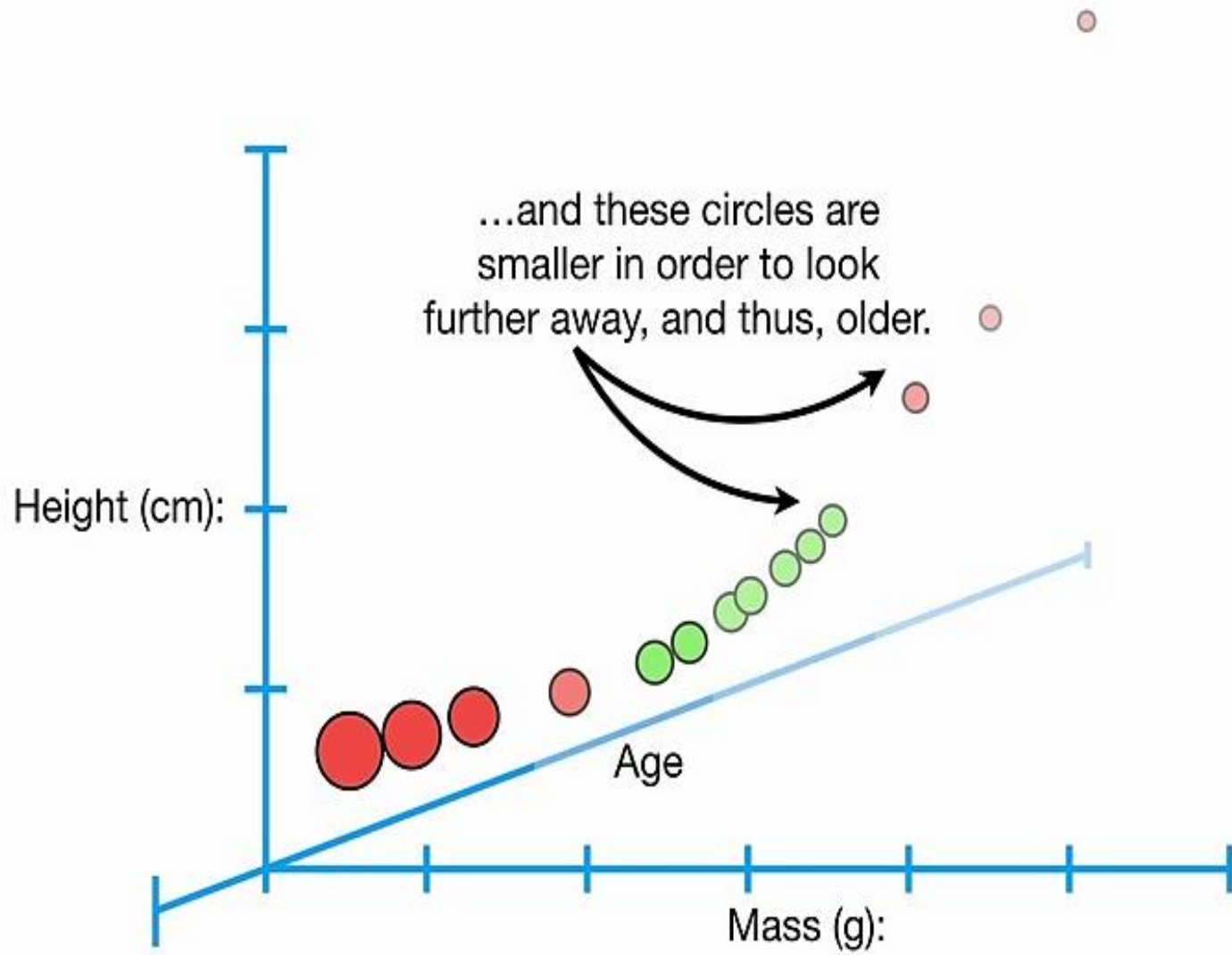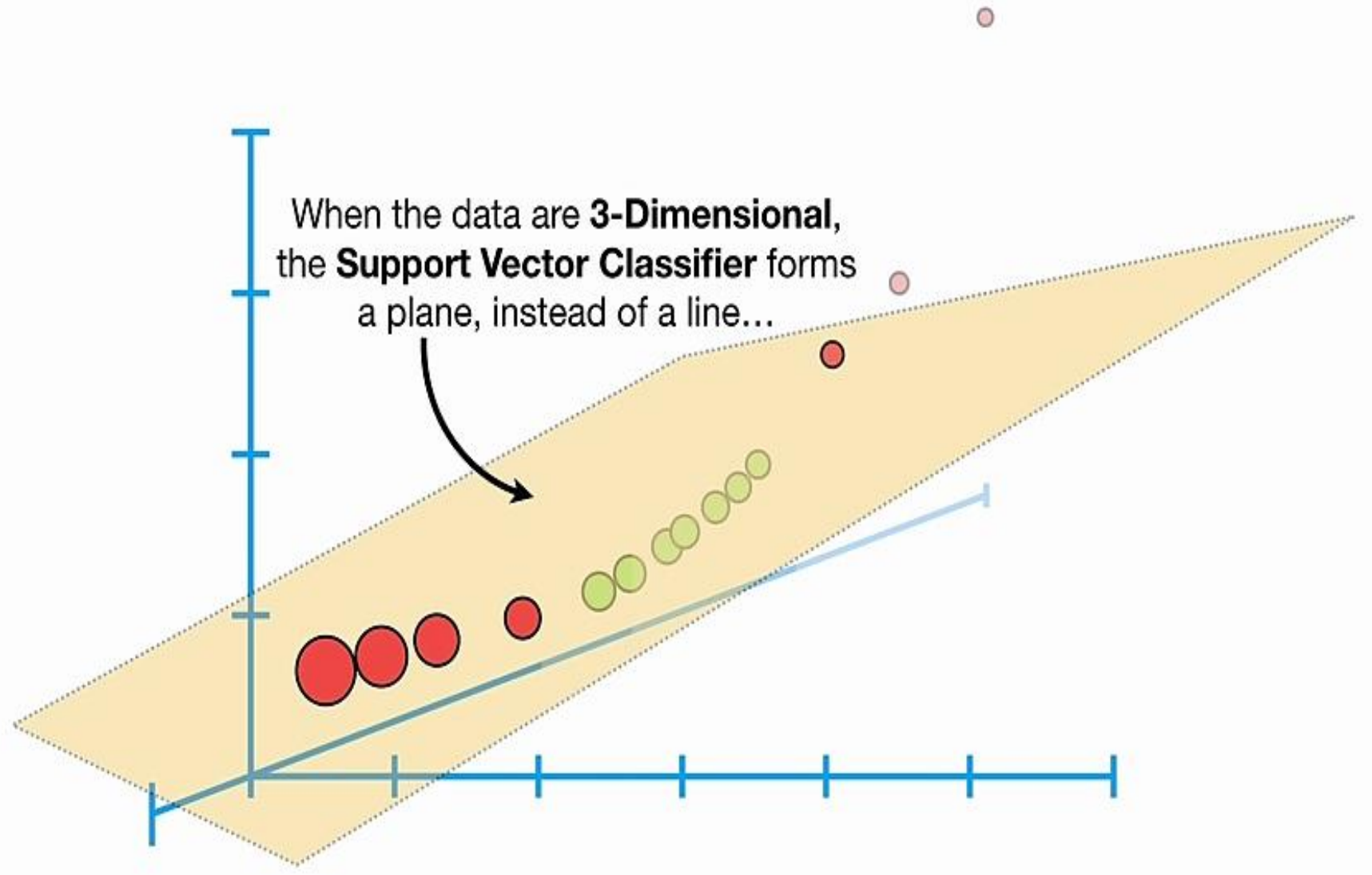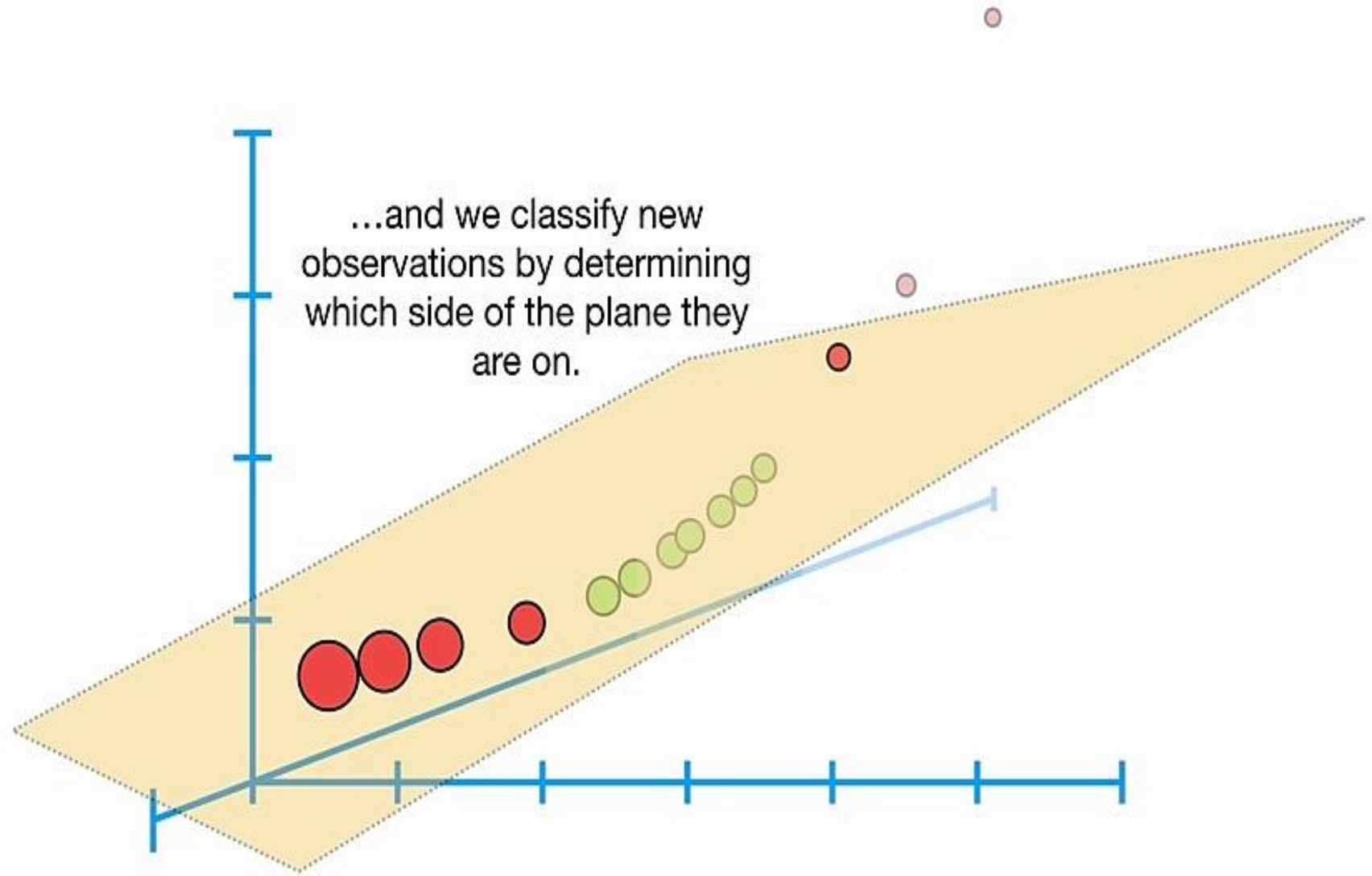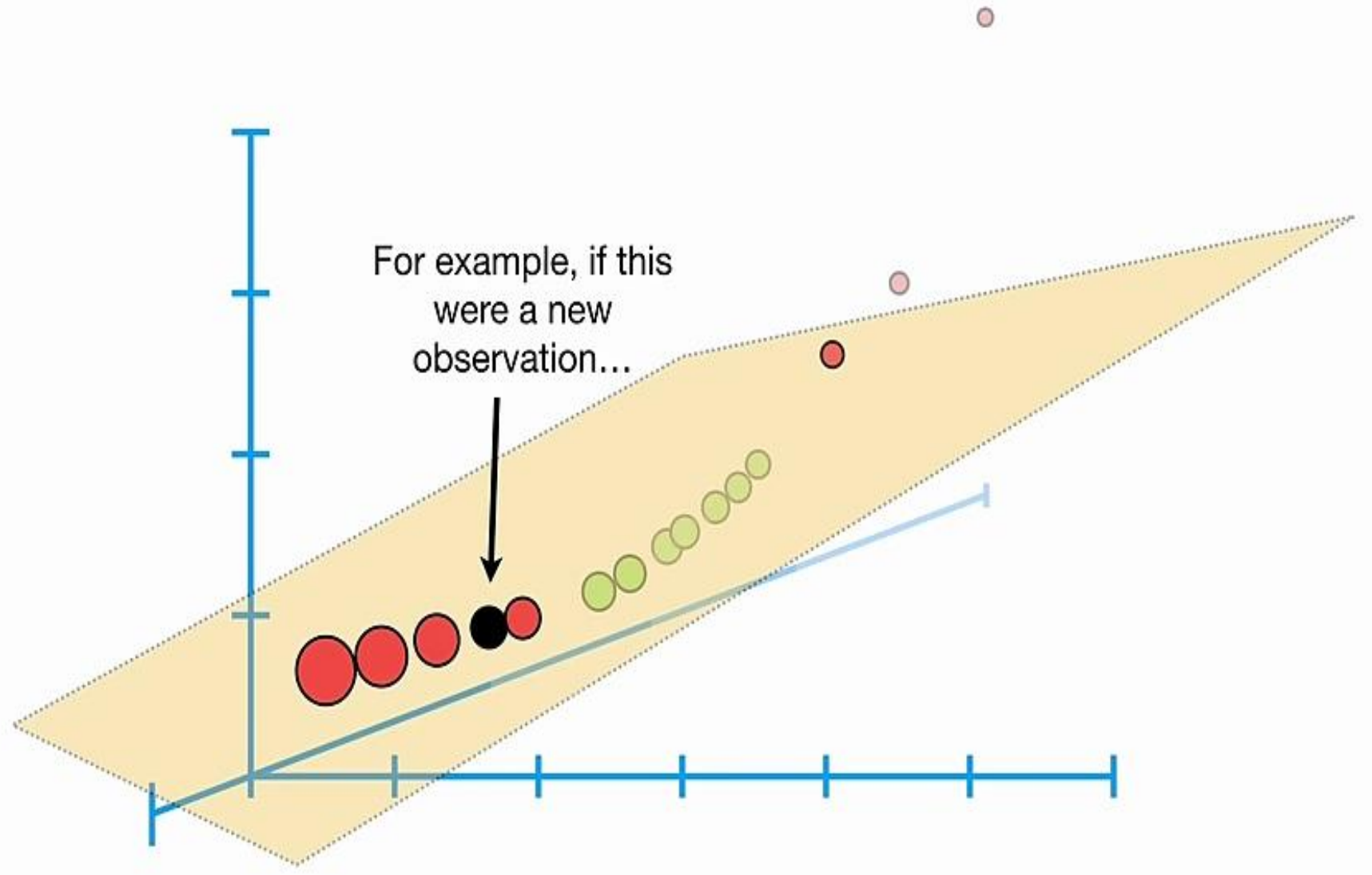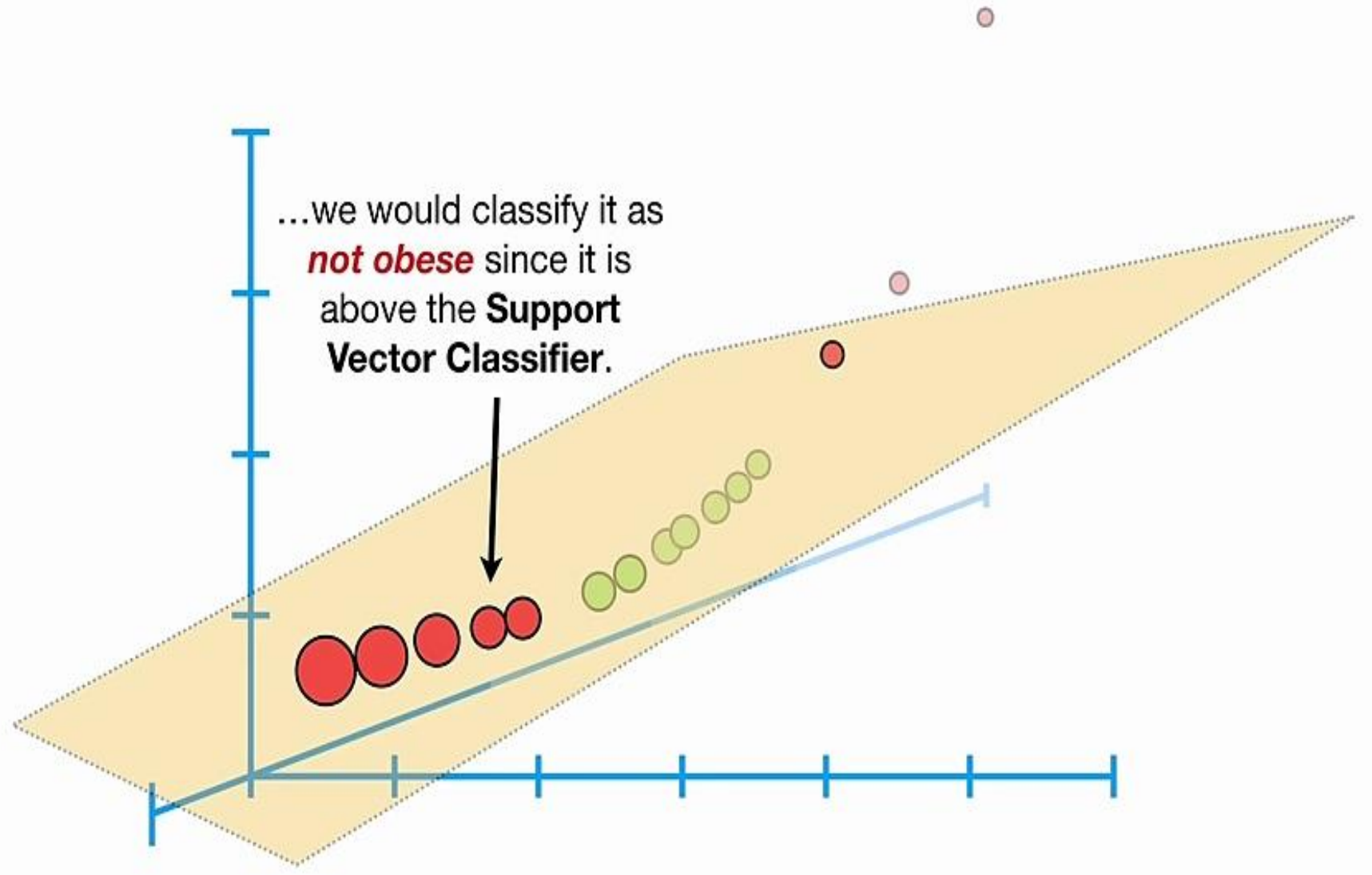
When the data are **3-Dimensional**, the **Support Vector Classifier** forms a plane, instead of a line…

...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**.
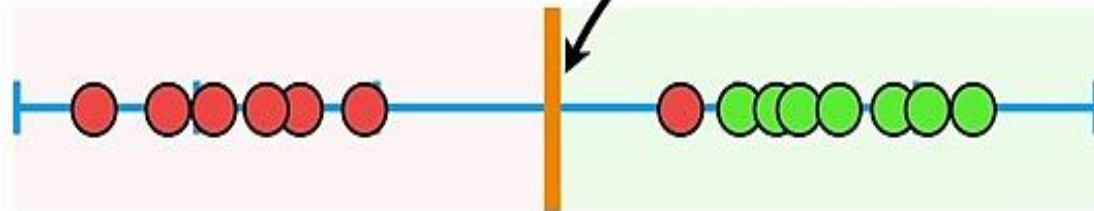
And when the data are in **2 Dimensions**, the **Support Vector Classifier** is a **1-Dimensional** *line* in a **2-Dimensional** space.
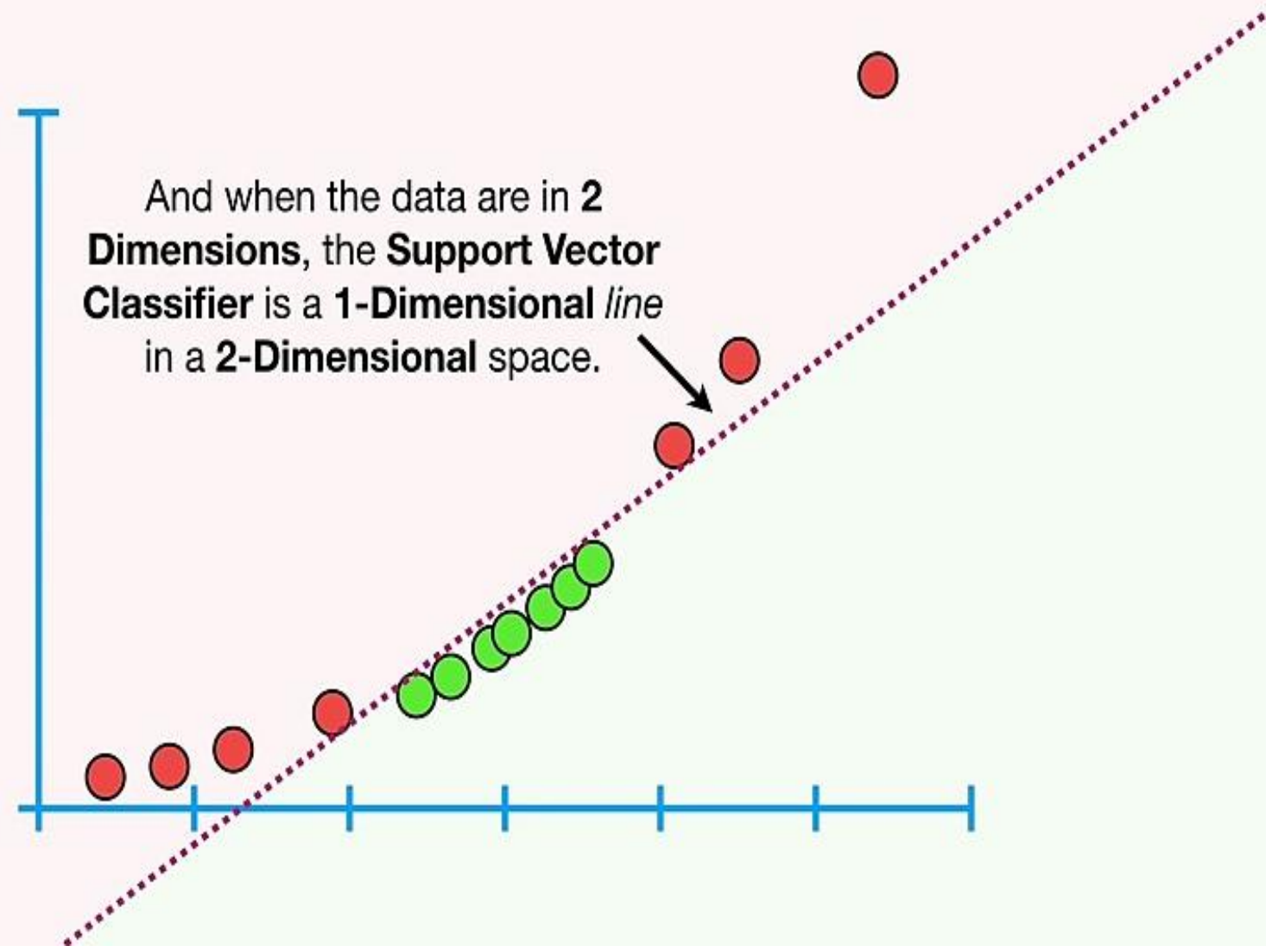
And when the data are **3-Dimensional**, the **Support Vector Classifier** is a **2-Dimensional** *plane* in a **3-Dimensional** space.
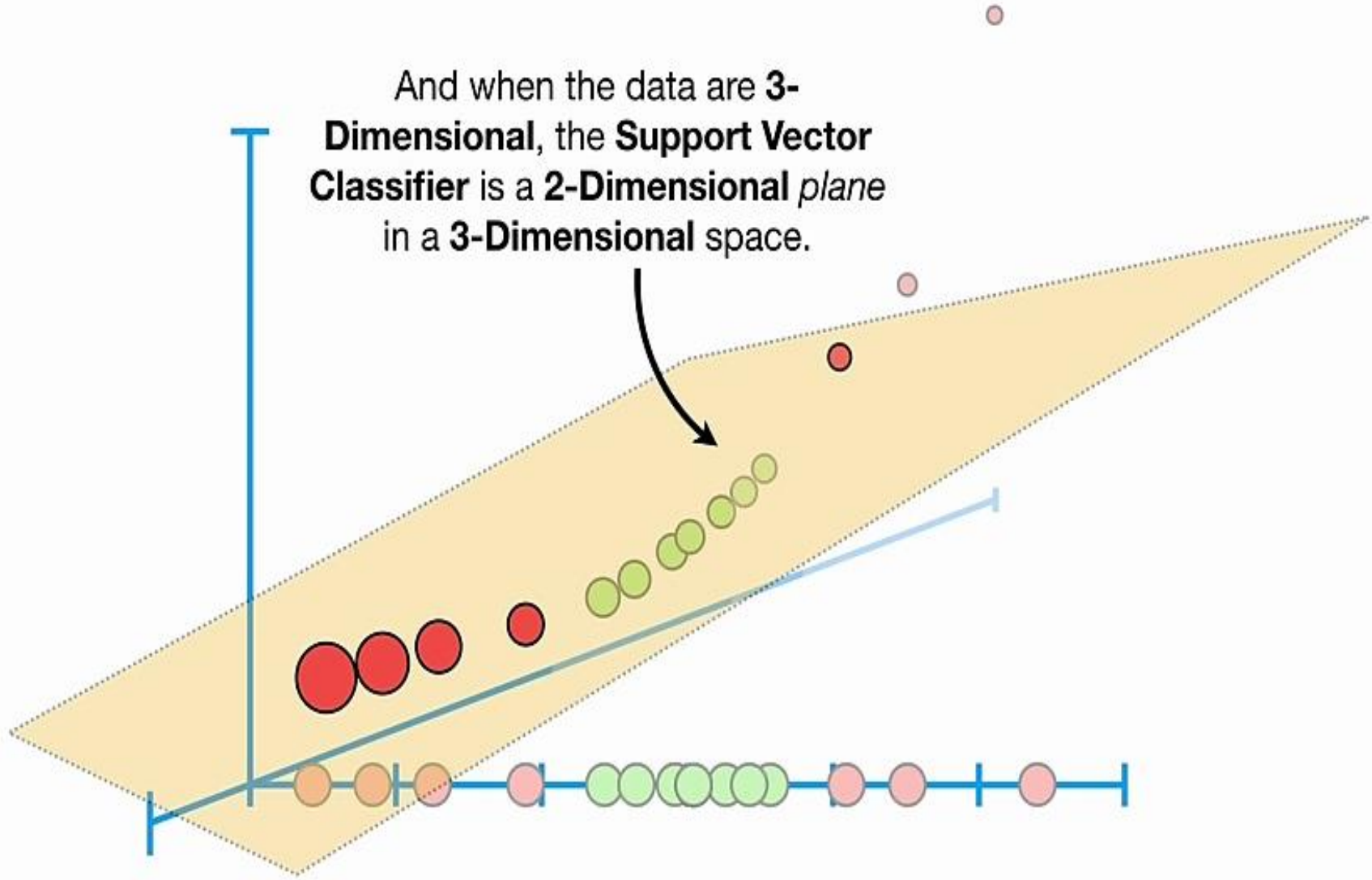
And when the data are in **4 or more Dimensions**, the **Support Vector Classifier** is a *hyperplane*.
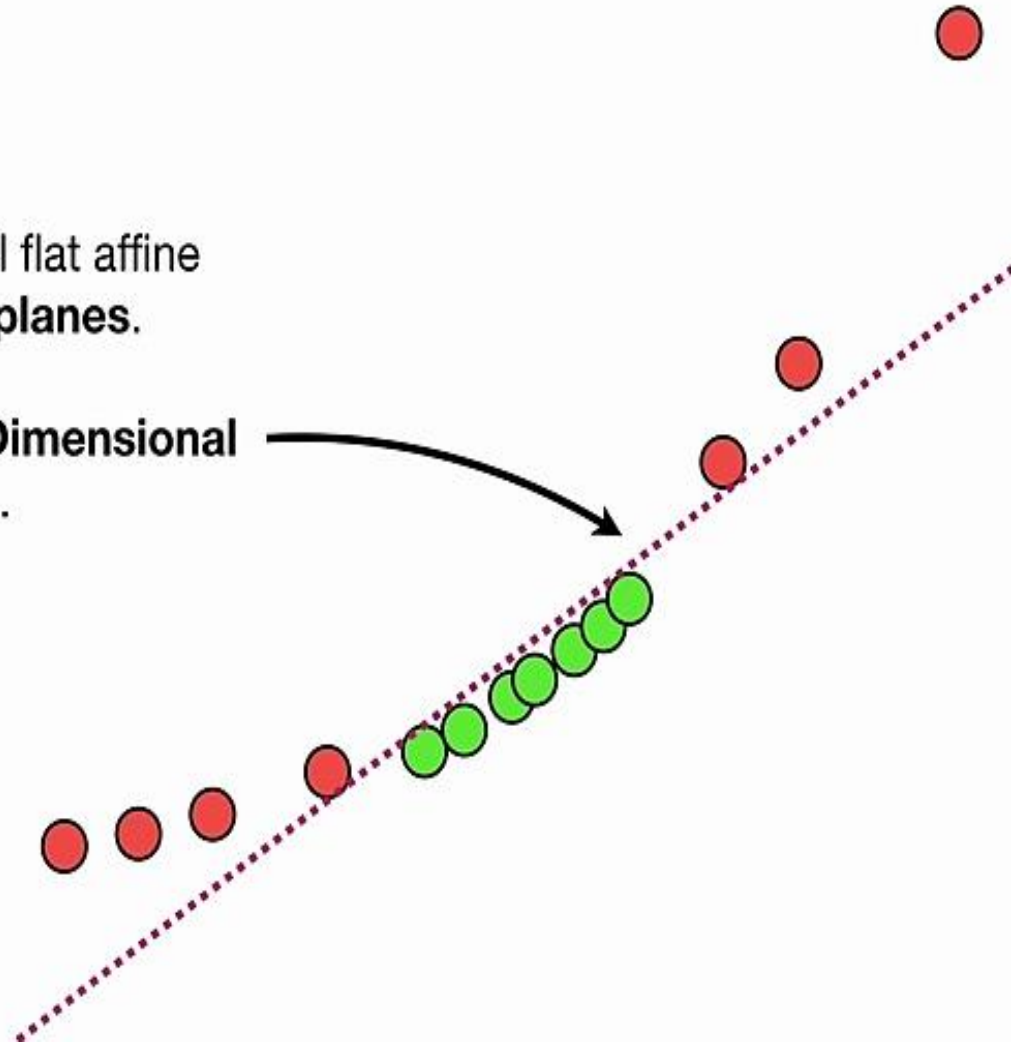
Non-linear SVM

Support Vector Classifiers seem pretty cool because they can handle…

Mass (g):
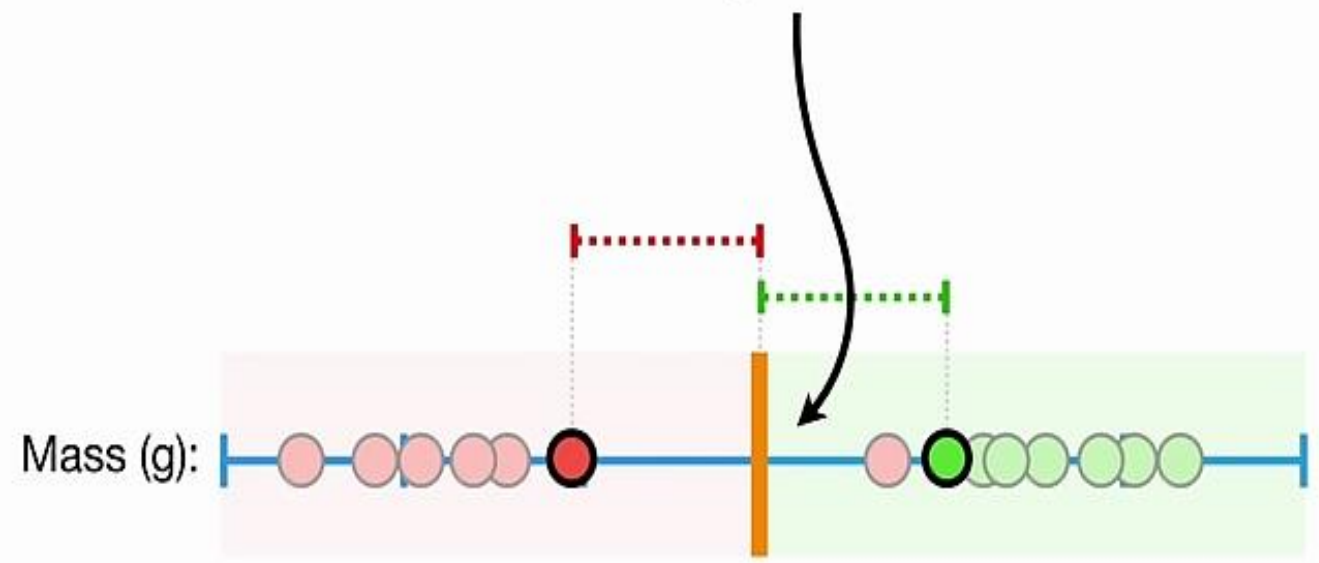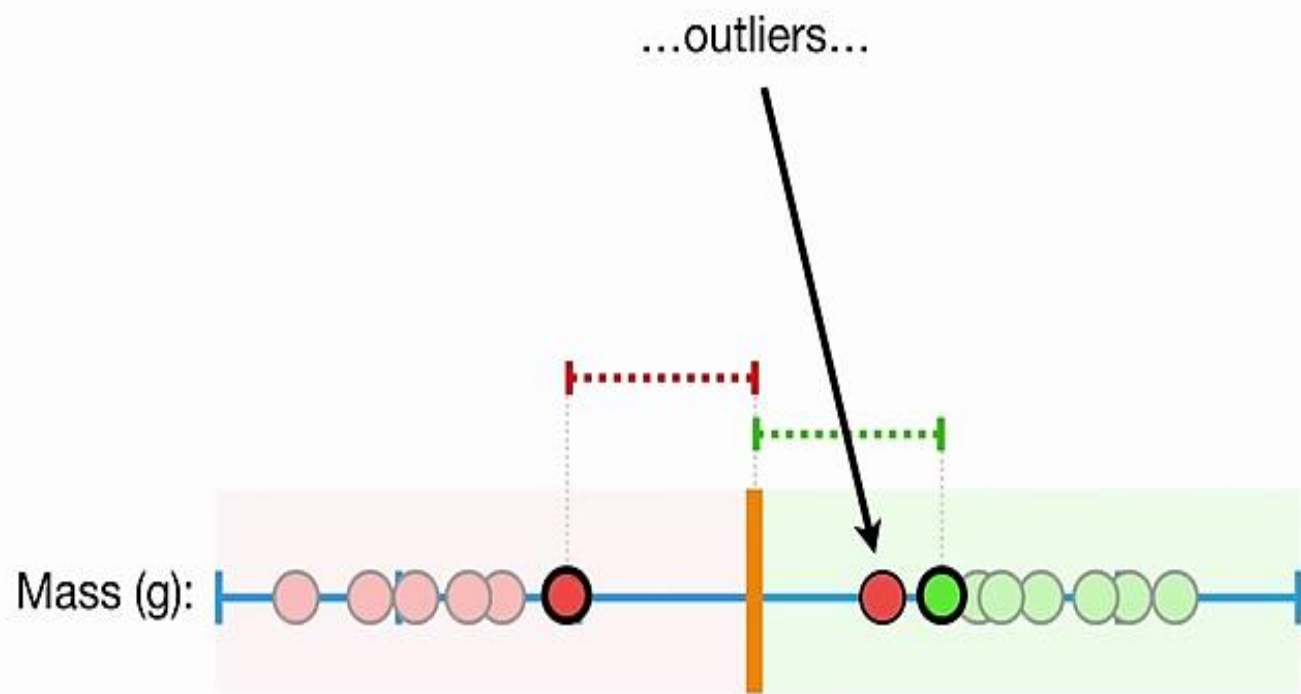
…and, because they allow misclassifications, they can handle overlapping classifications…

Mass (g):
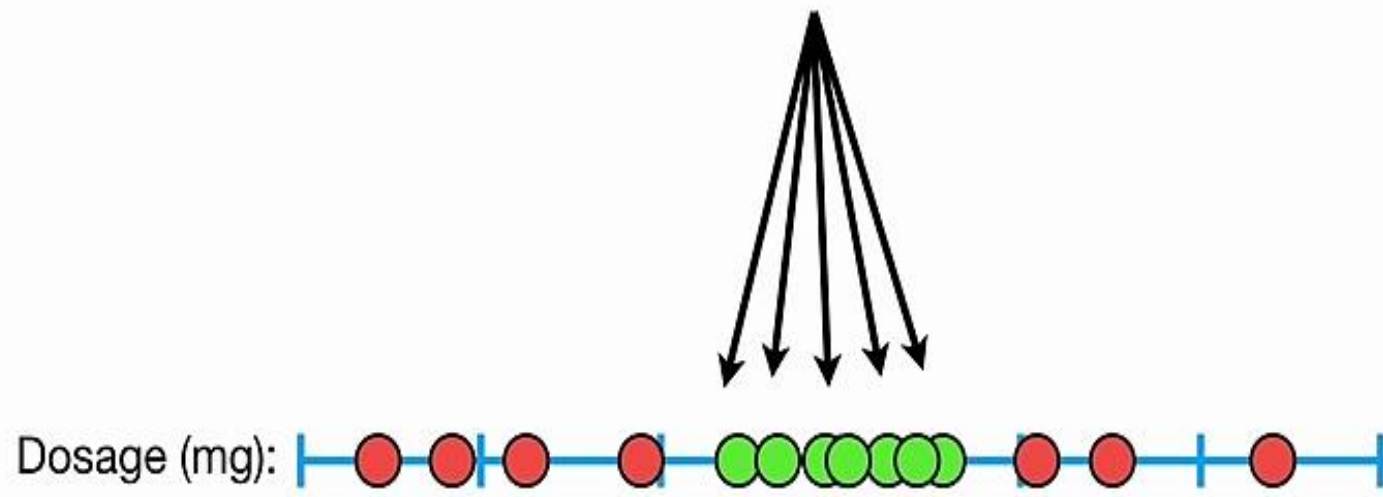
Can we do better than **Maximal Margin Classifiers** and **Support Vector Classifiers**?

Dosage (mg):

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

y-axis

Dosage (mg):

...the **y-axis** value = **Dosage²**

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 classify the observation as *not cured* because it ended up on this side of the **Support Vector Classifier**.

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):

1) Start with data in a relatively low dimension…

y-axis

Dosage (mg):

3) Find a **Support Vector Classifier** that separates the higher dimensional data into two groups.

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):

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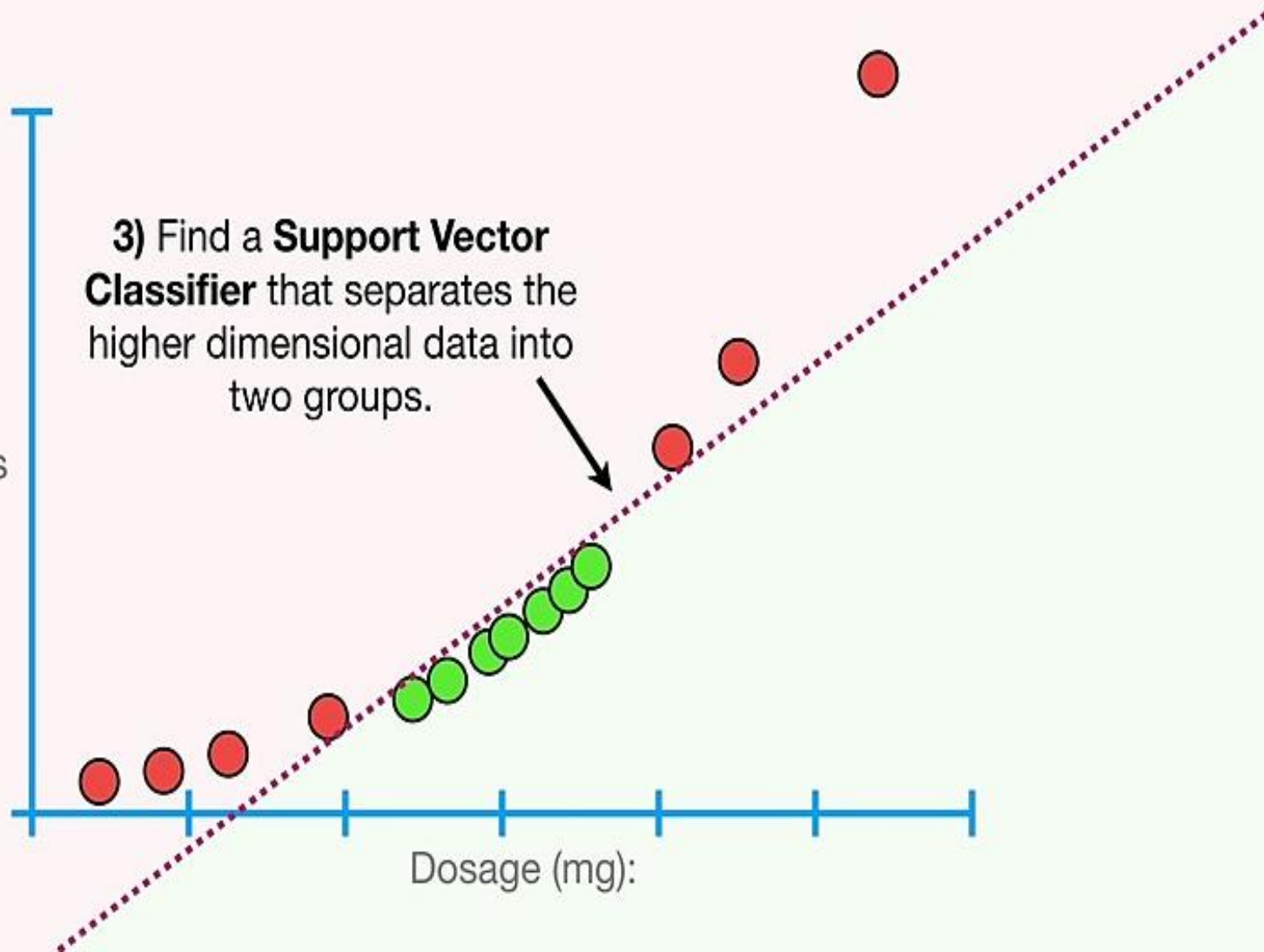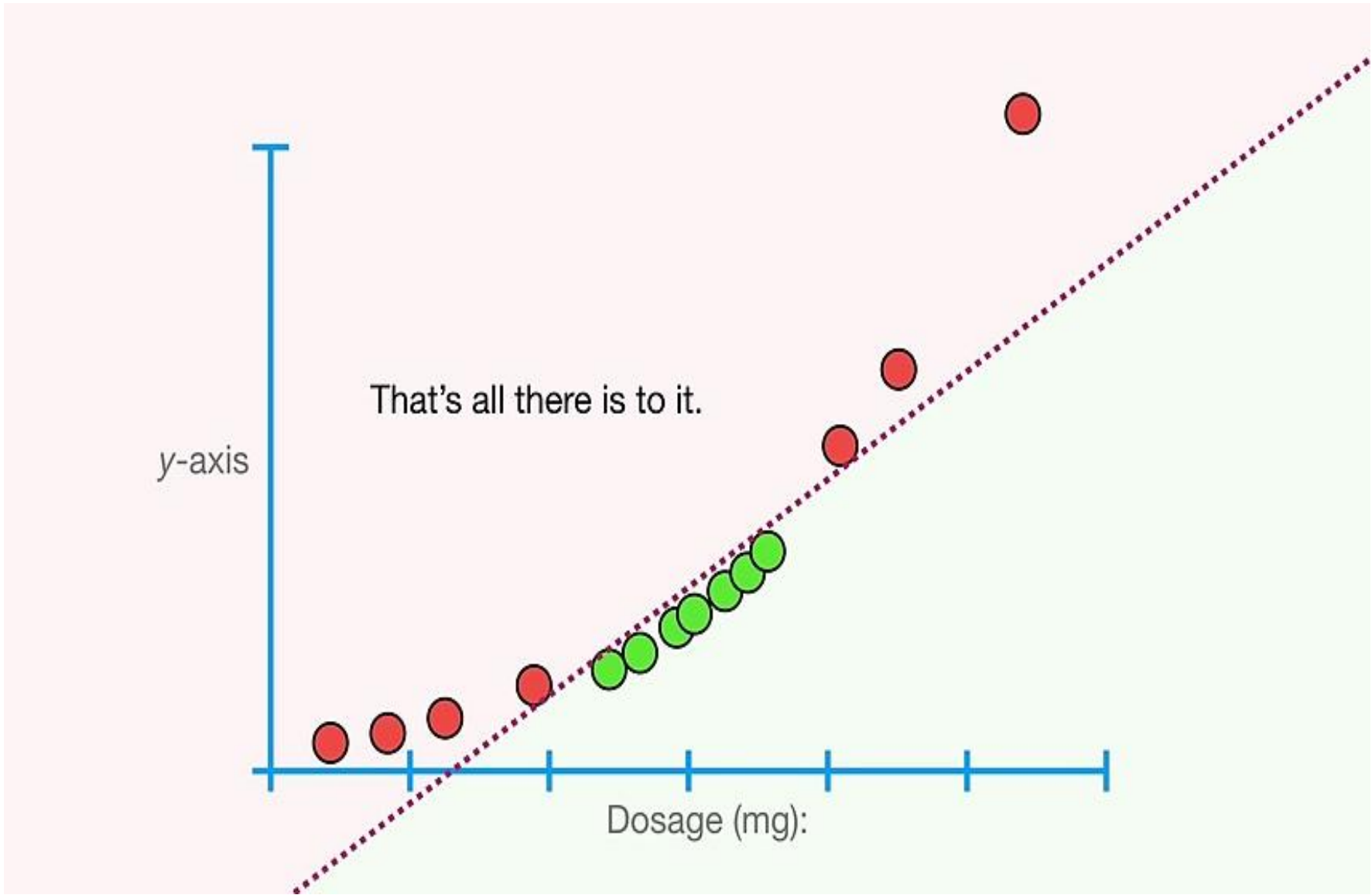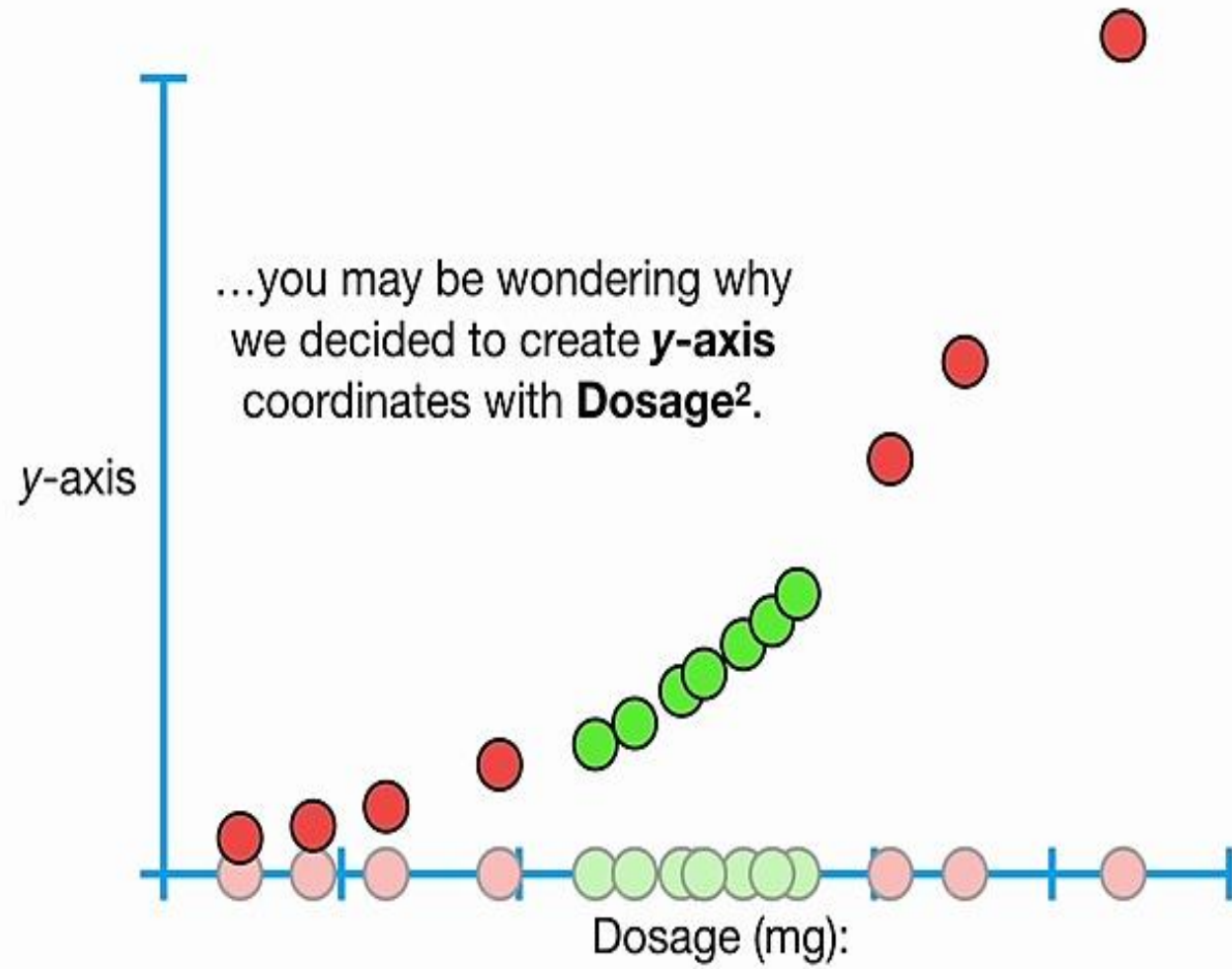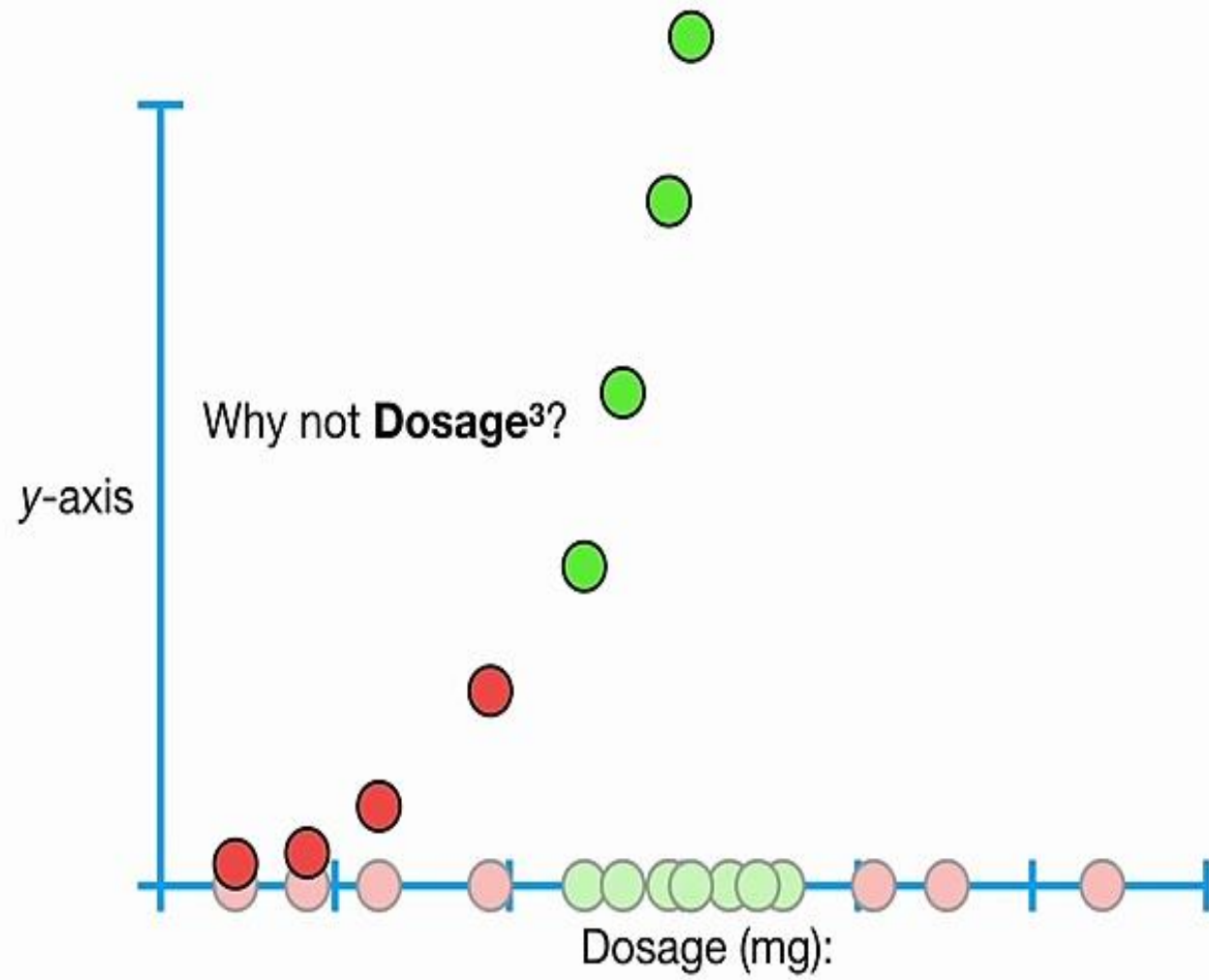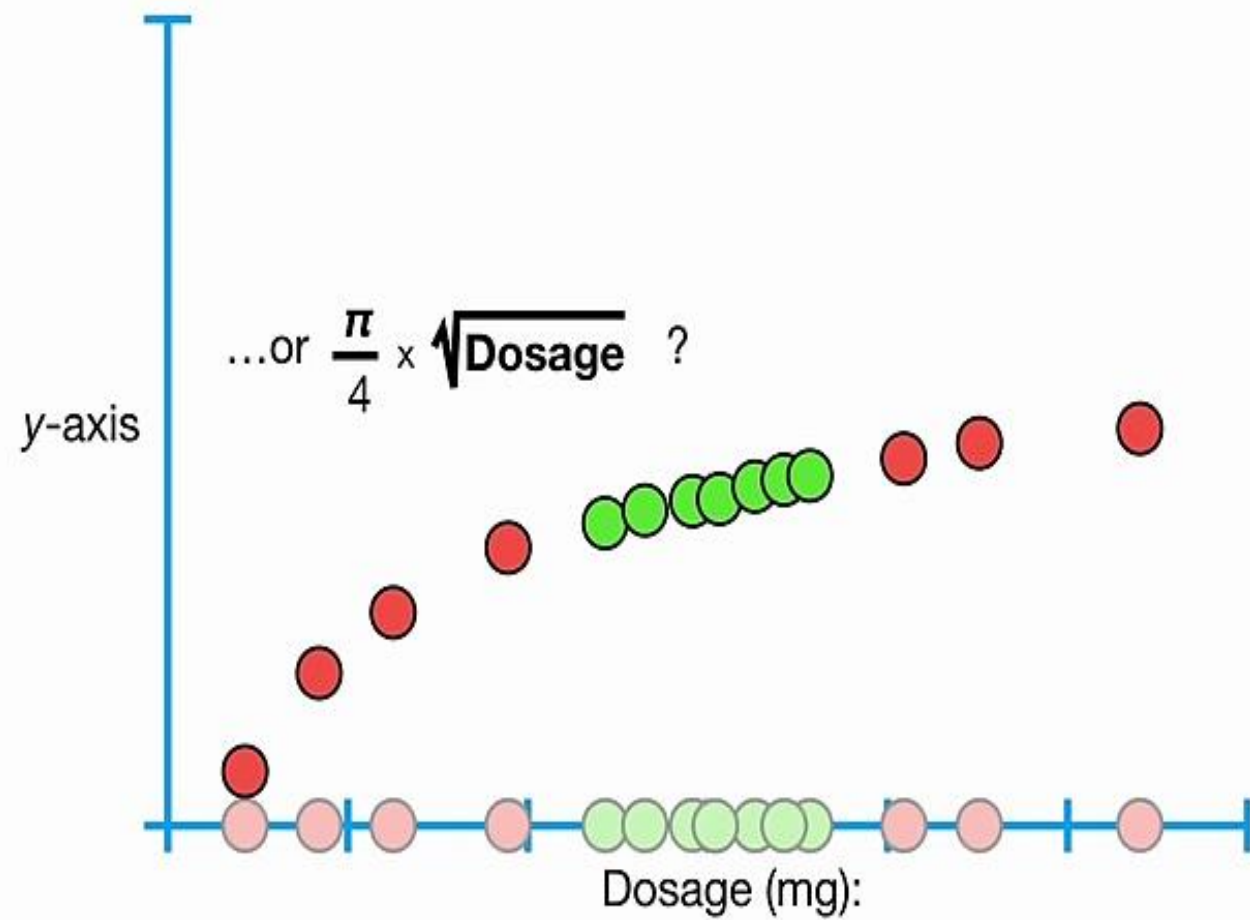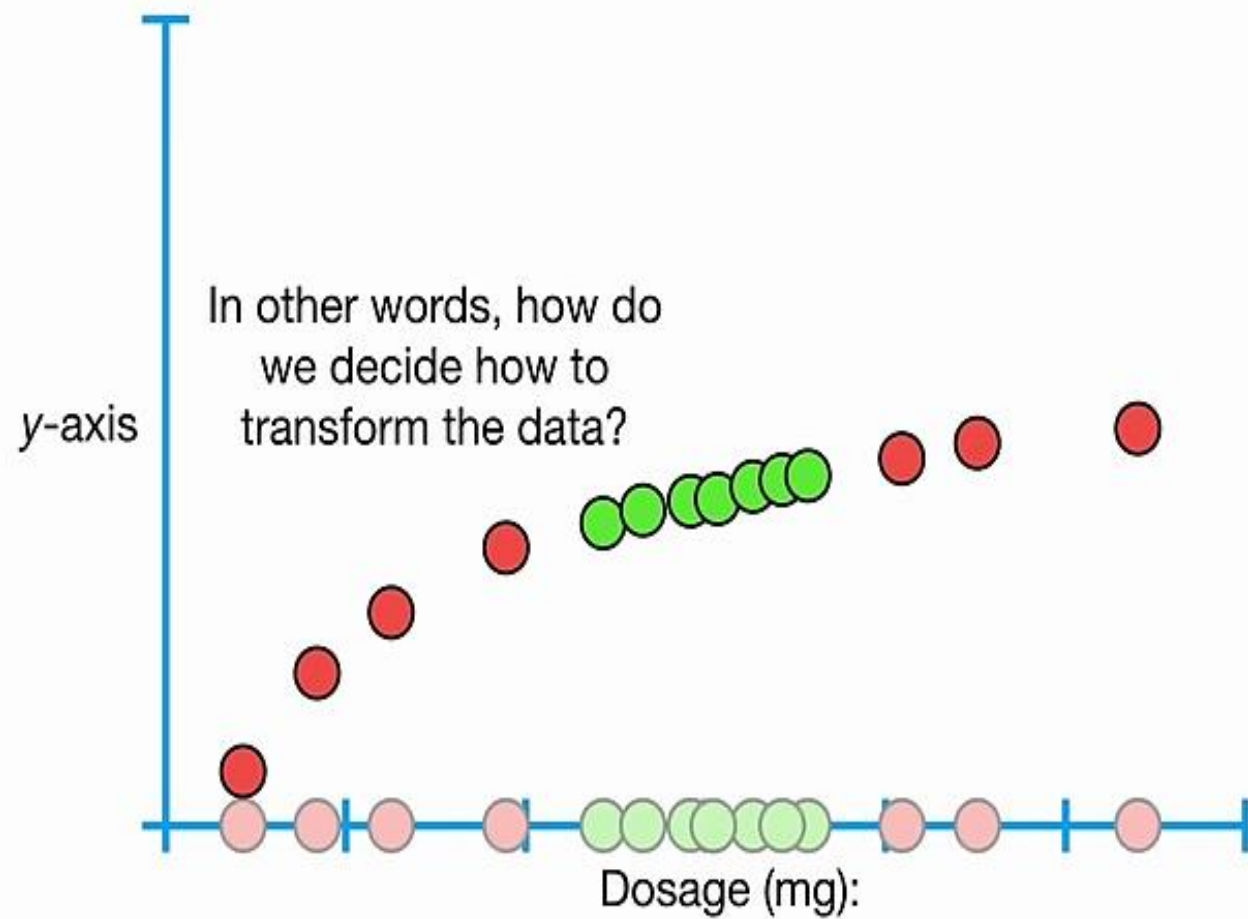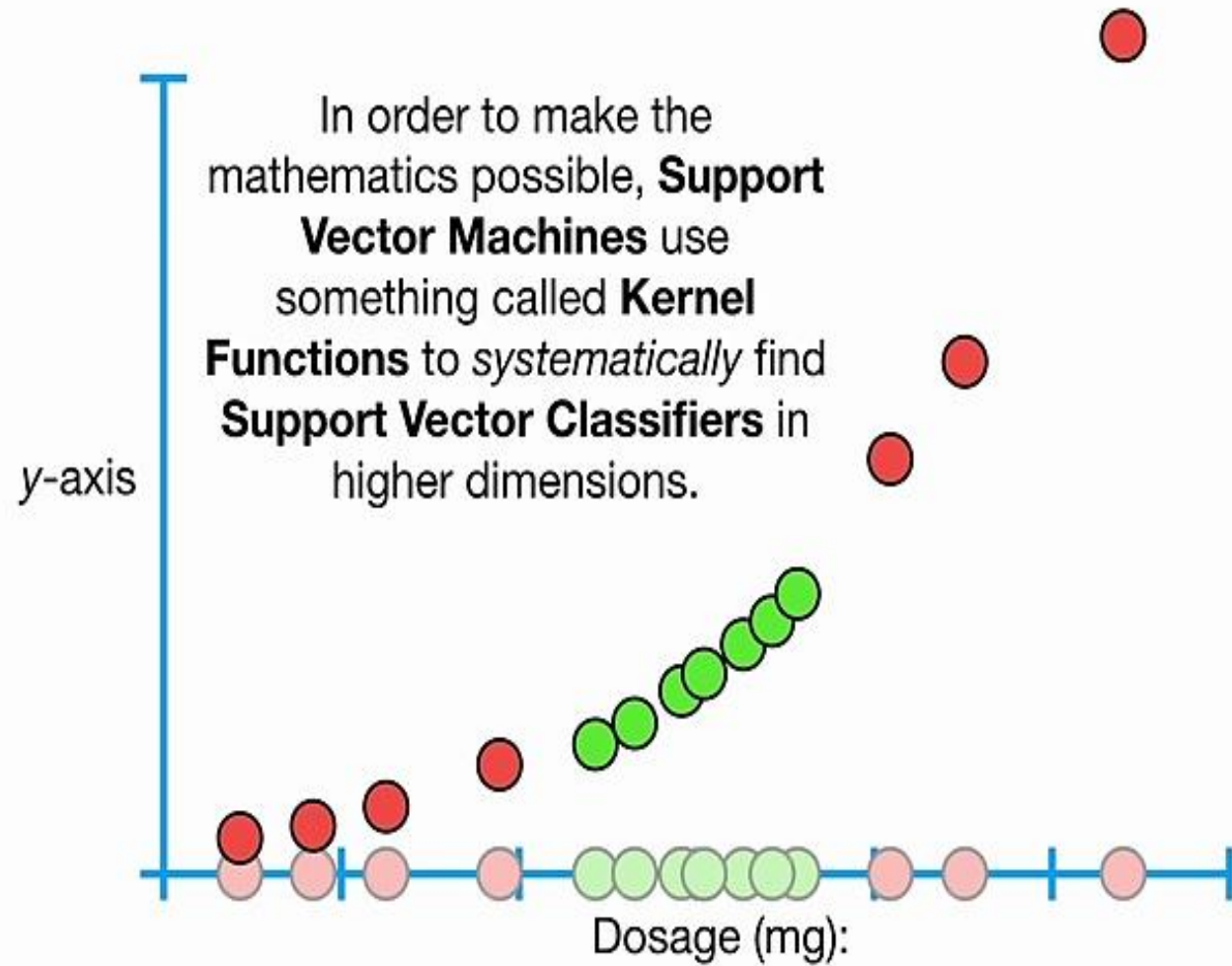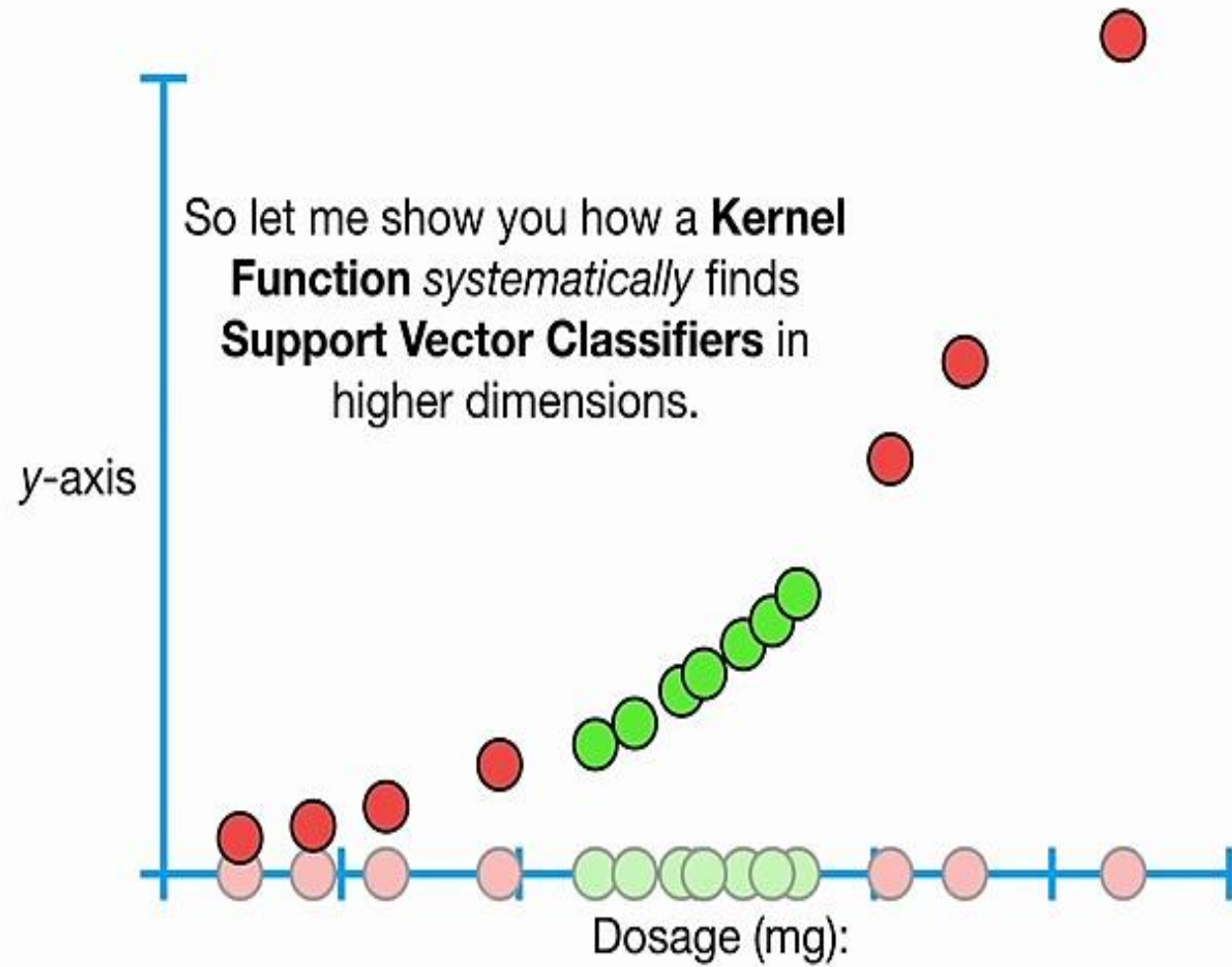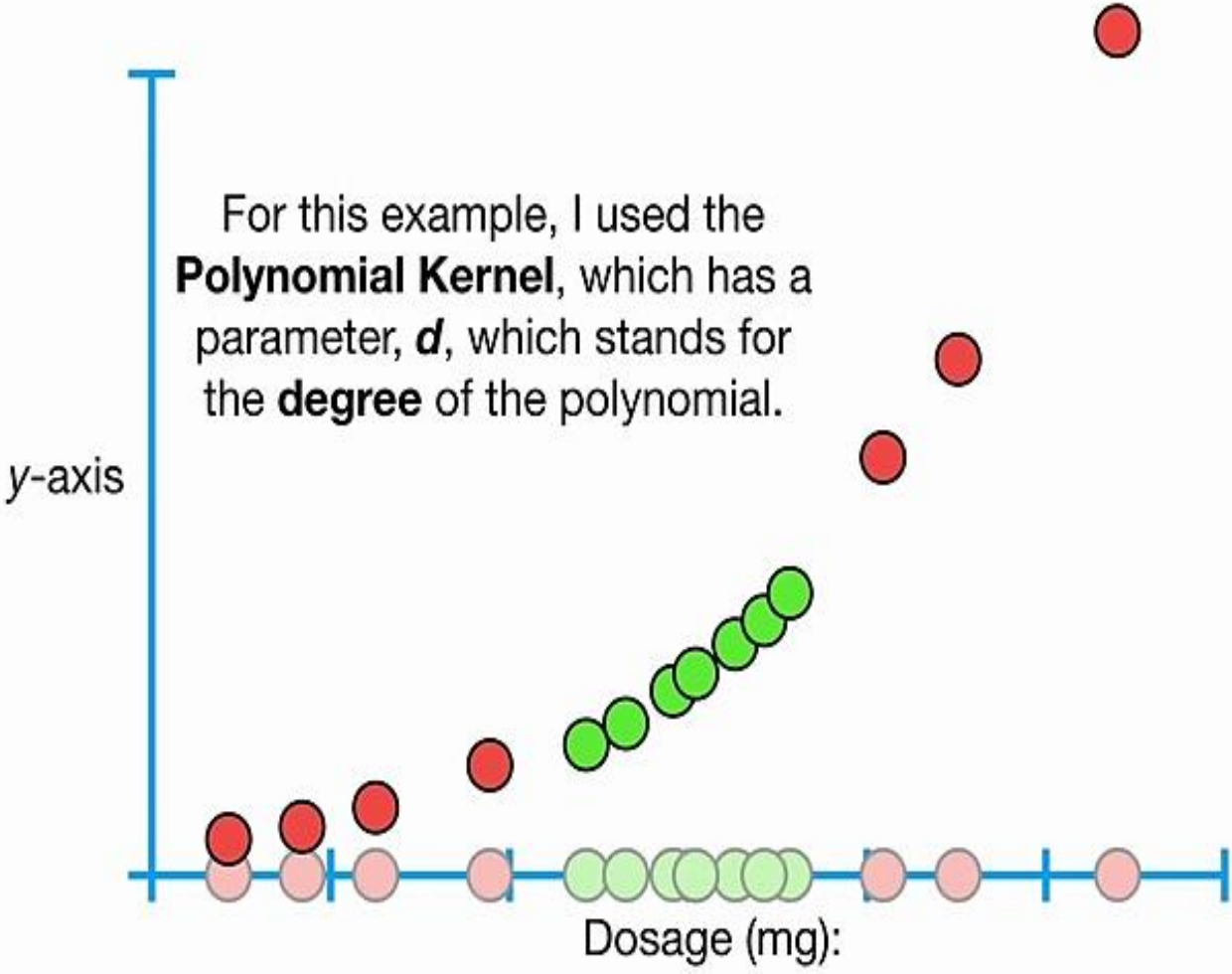
When **d = 1**, the **Polynomial Kernel** computes the relationships between each pair of observations in **1-Dimension**…

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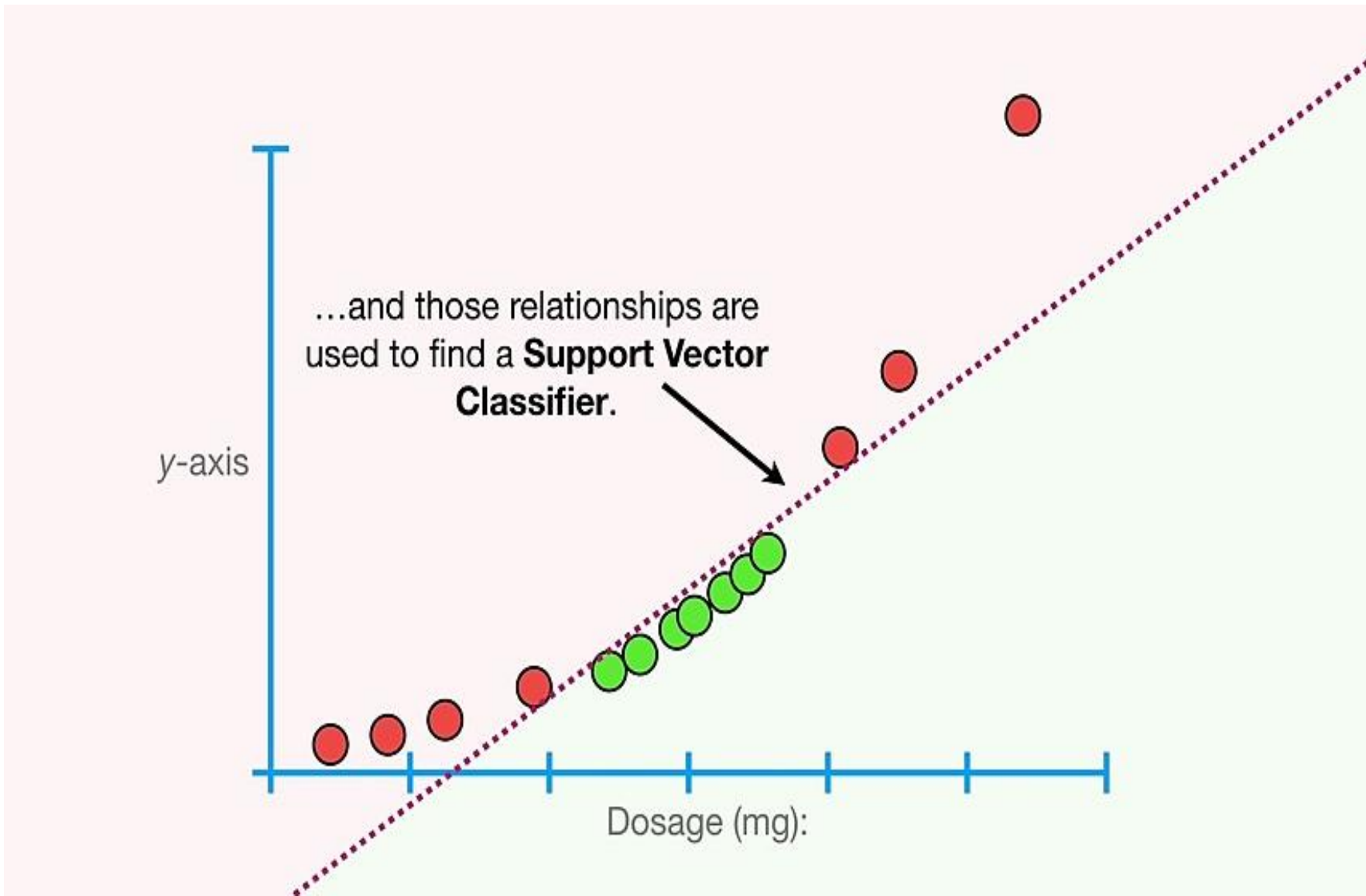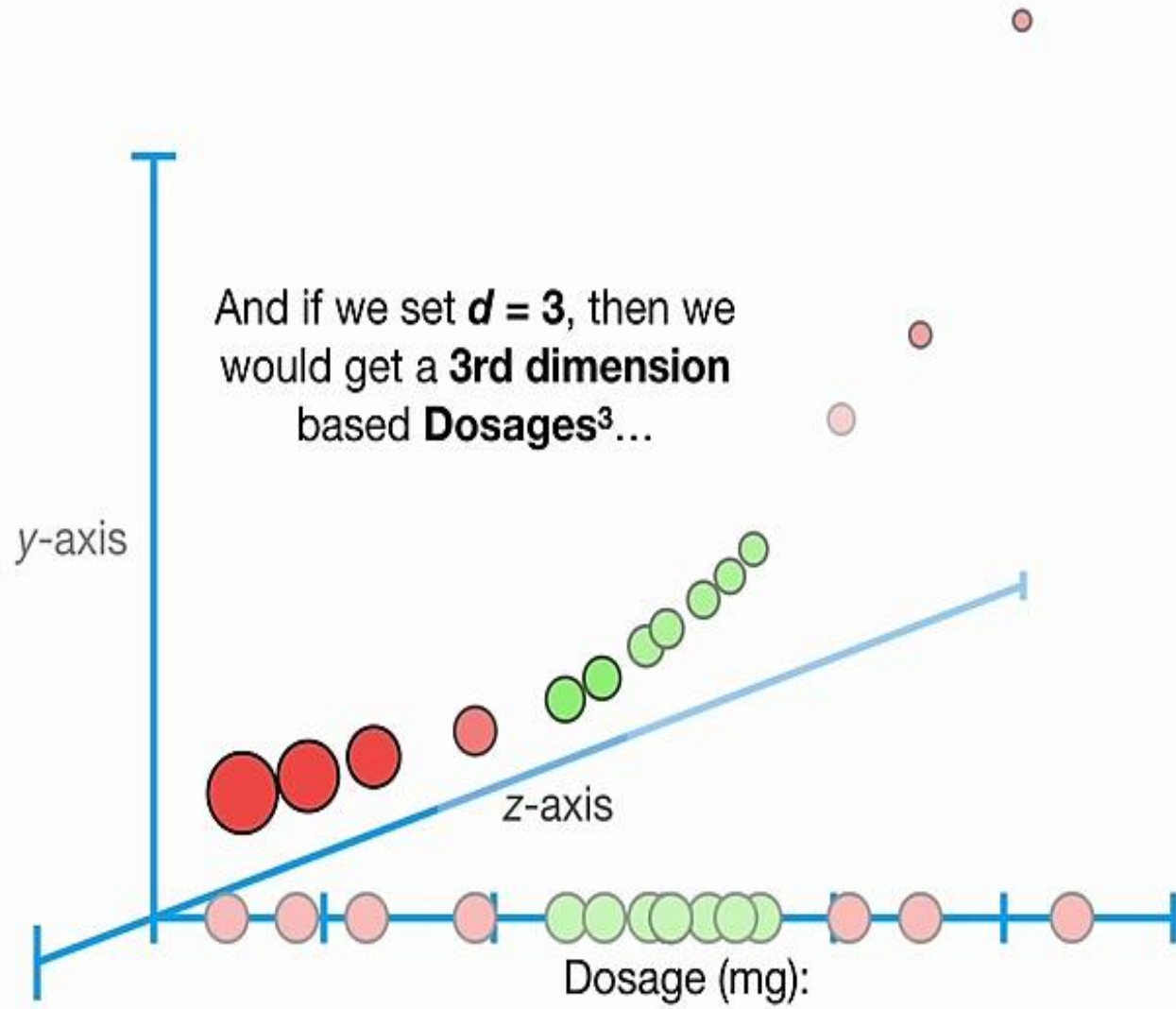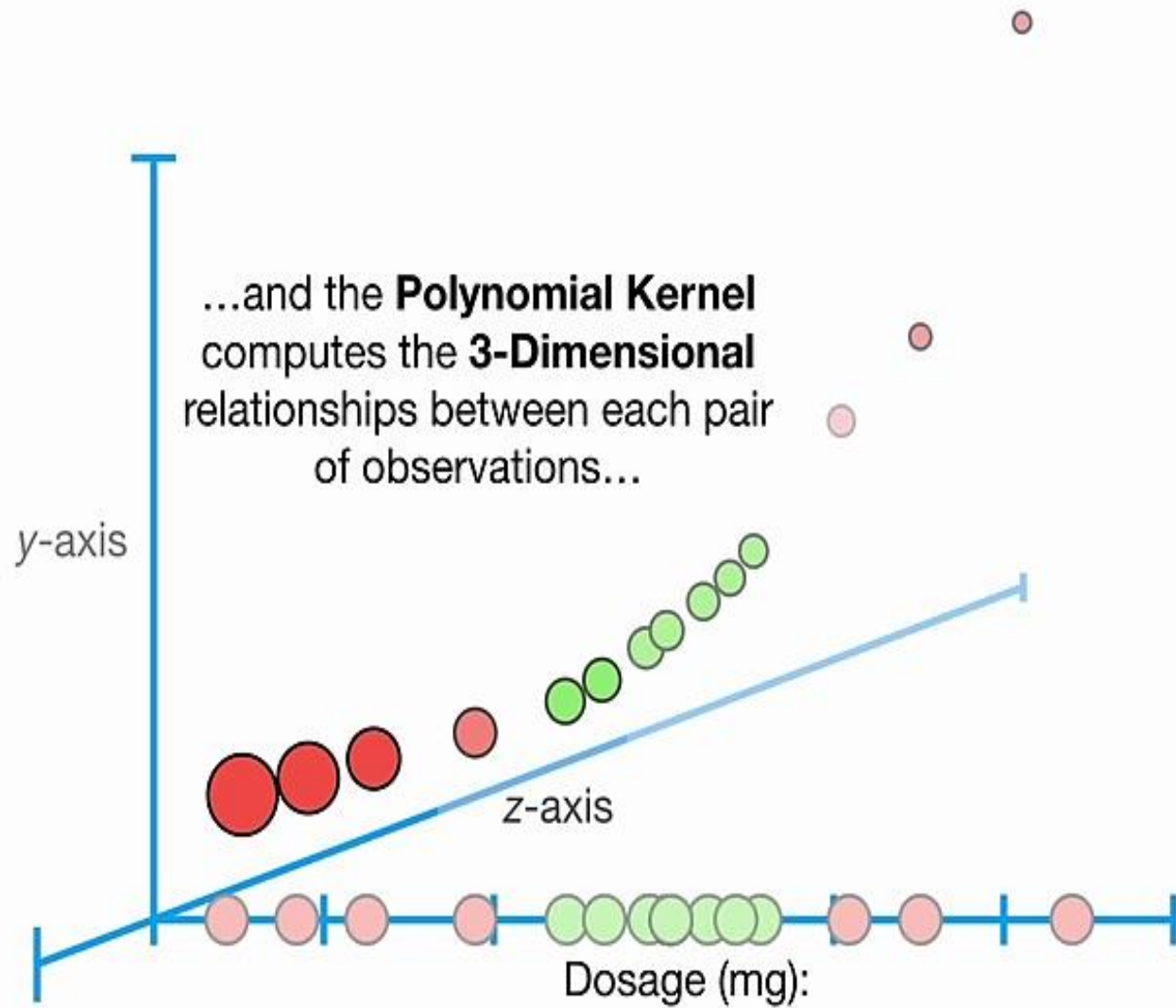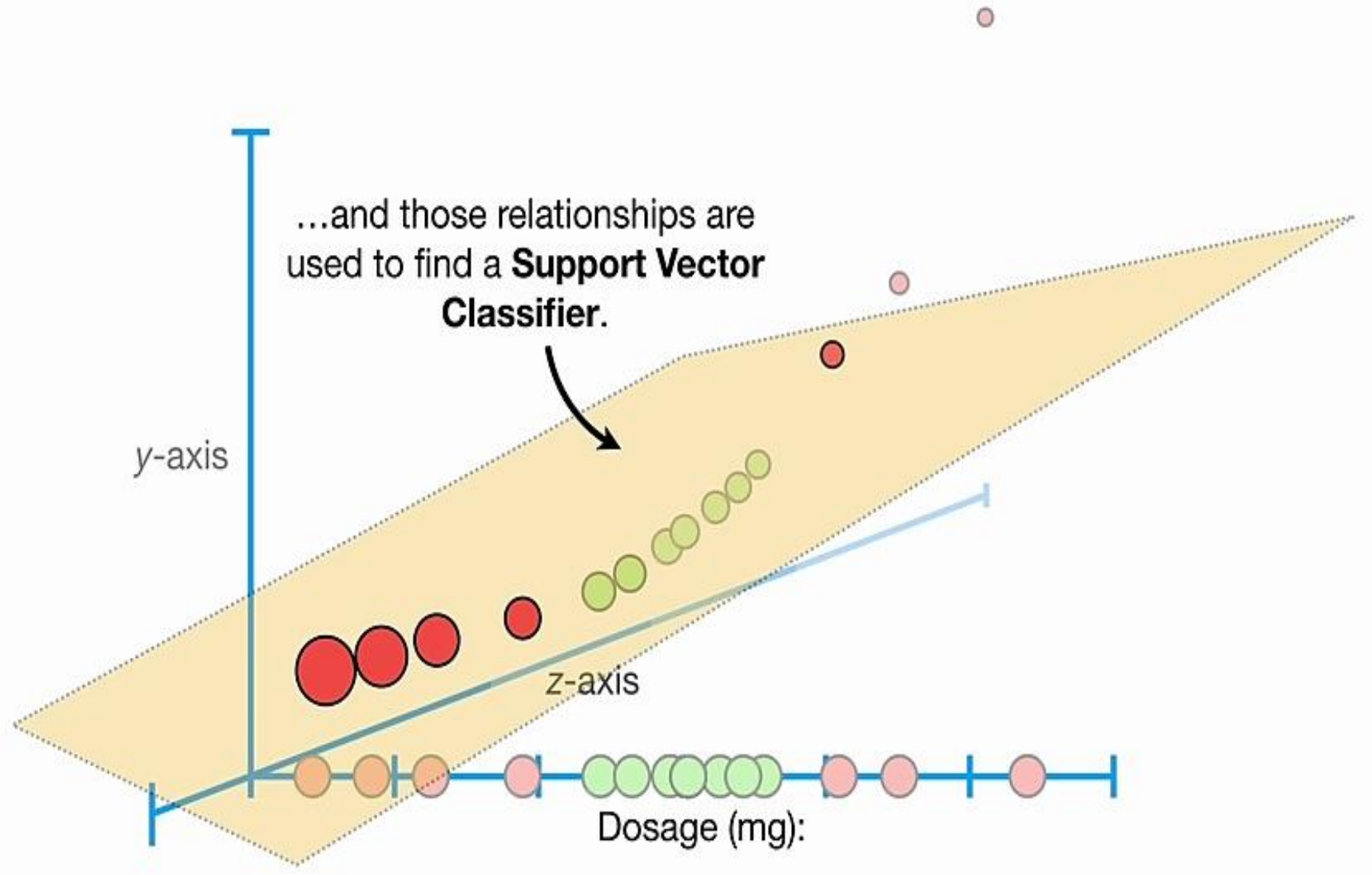
And when d = **4 or more**, then we get even more dimensions to find a **Support Vector Classifier.**

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

$d = 1$

Dosage (mg):

...and the relationships between each pair of observations are used to find a **Support Vector Classifier**.

y-axis

z-axis

Dosage (mg):
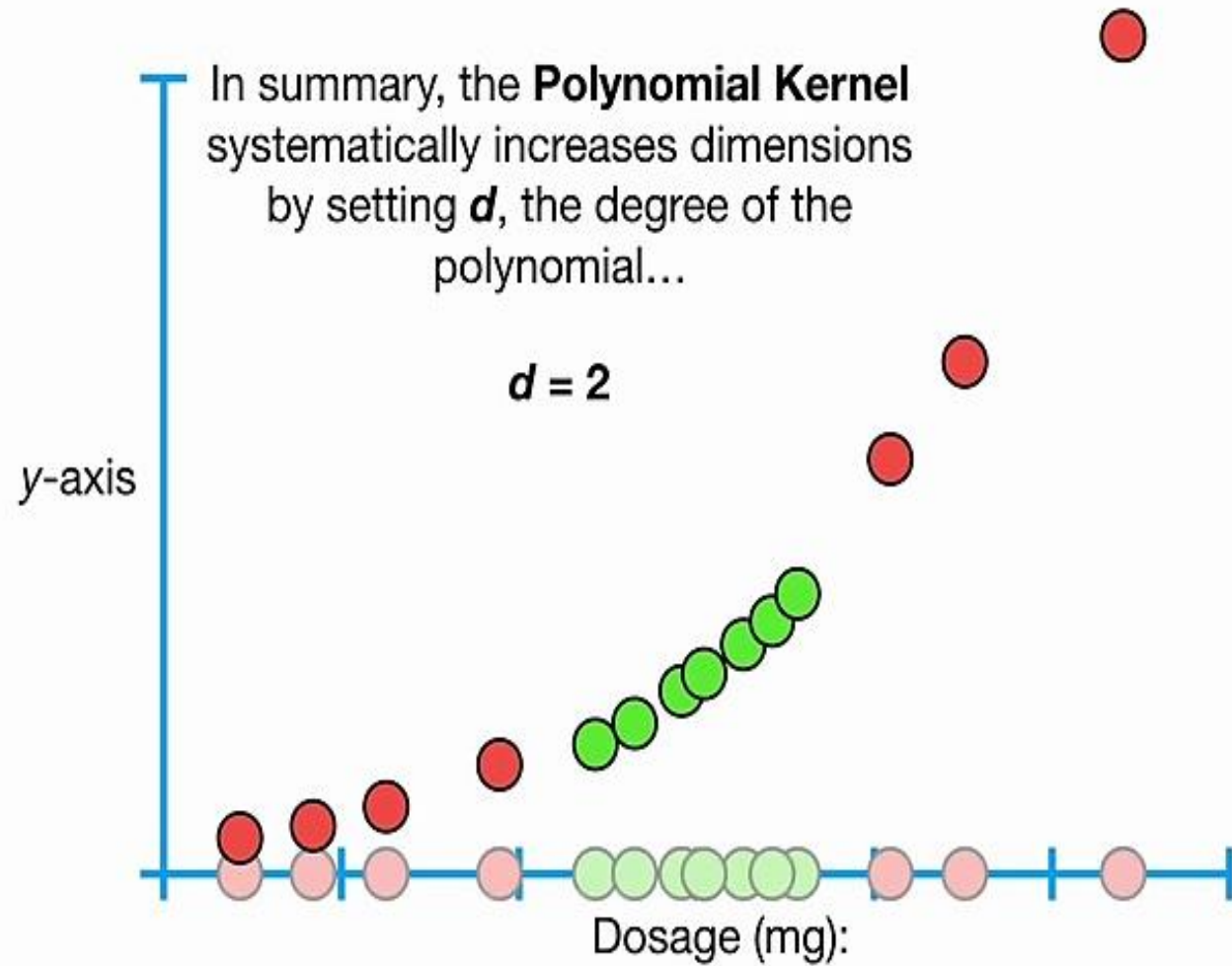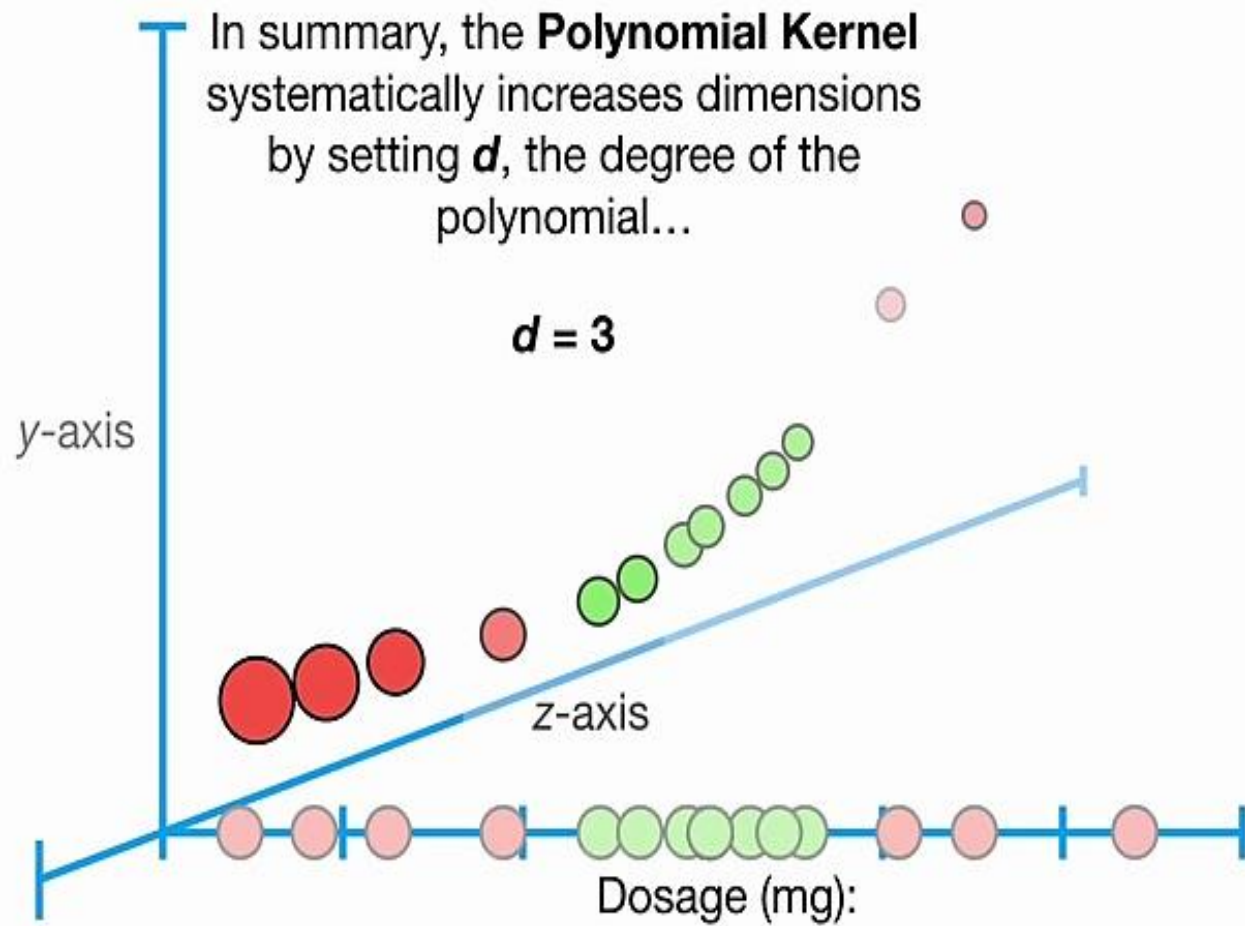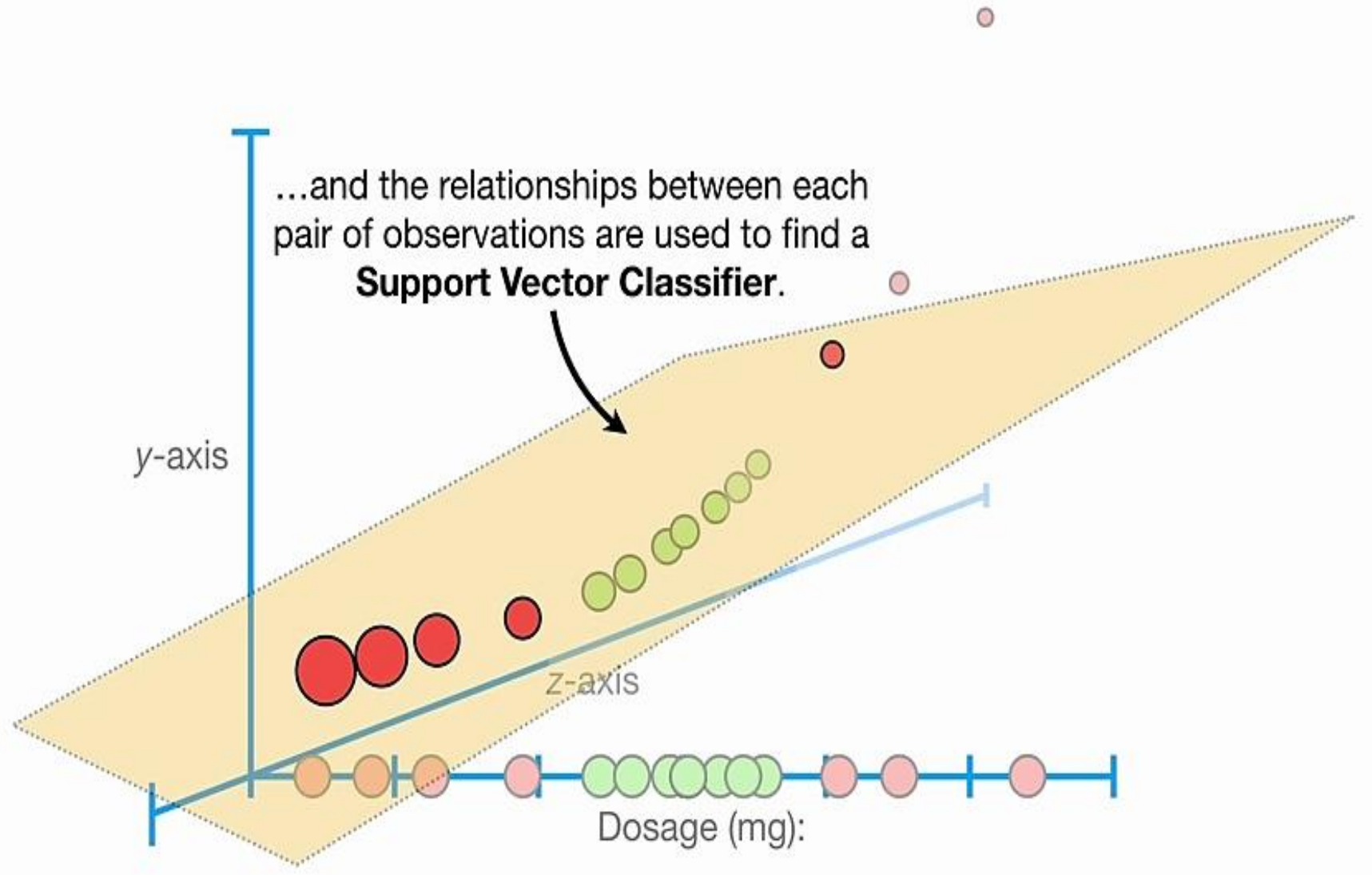
Last but not least, we can find a good value for *d* with **Cross Validation**.
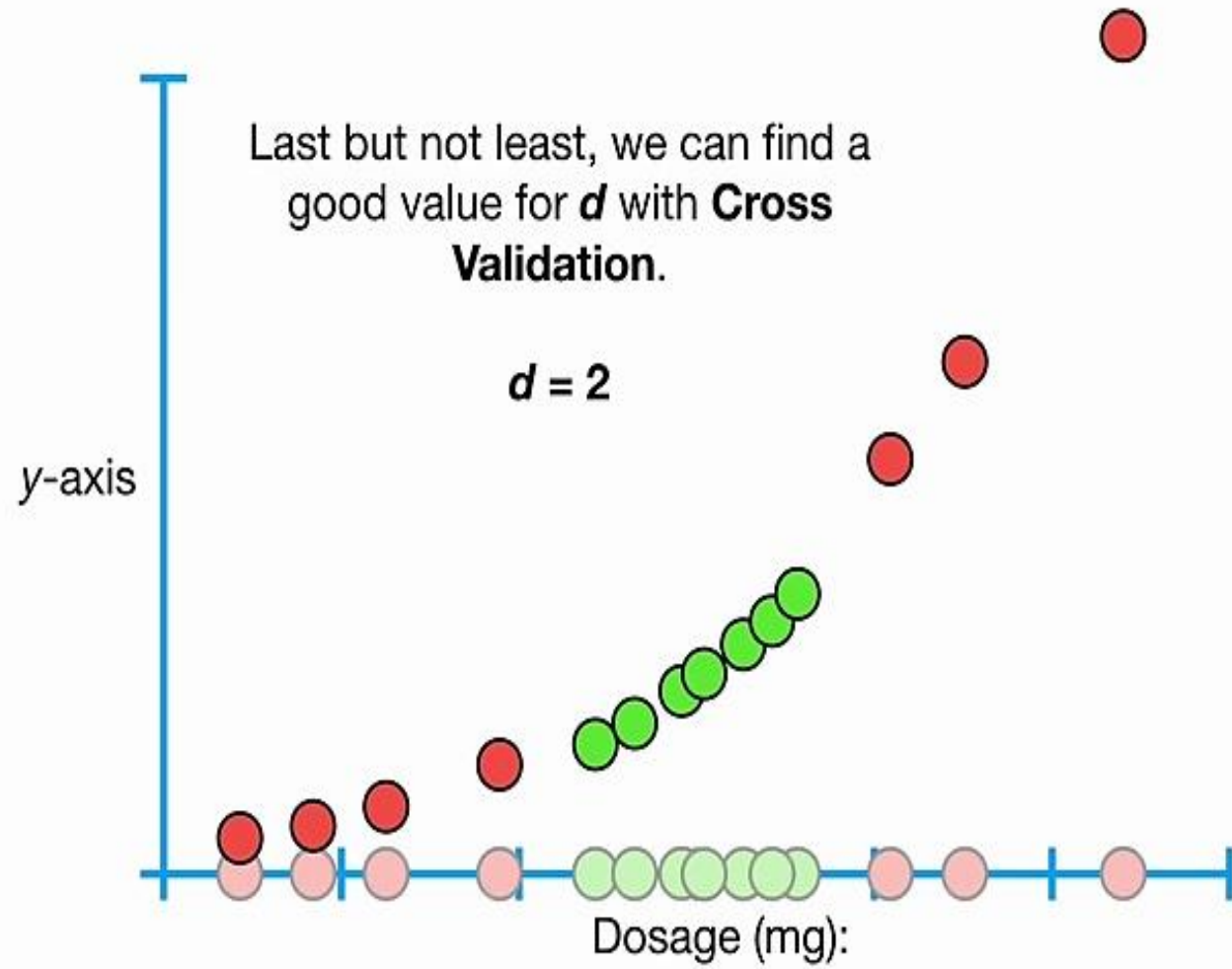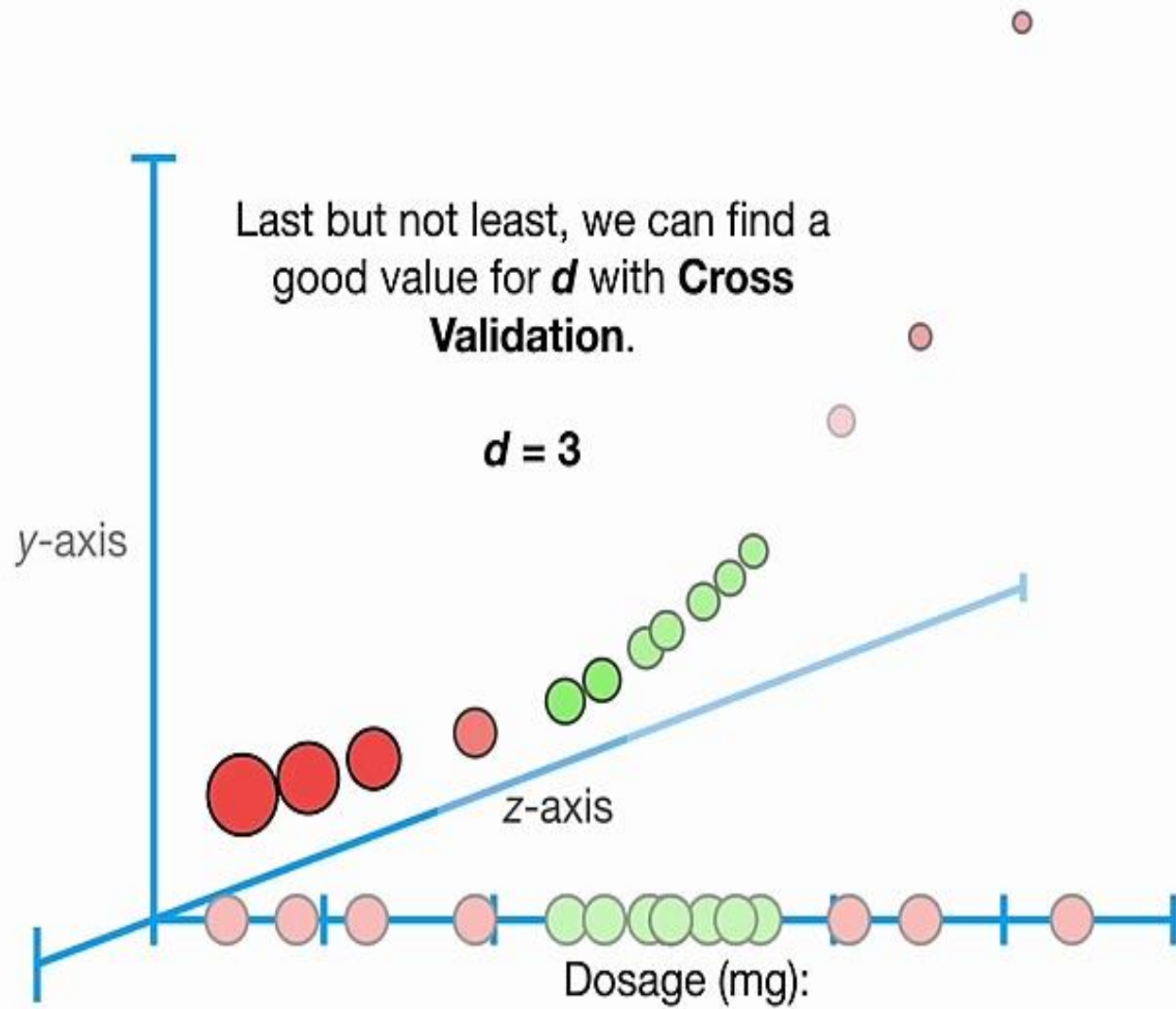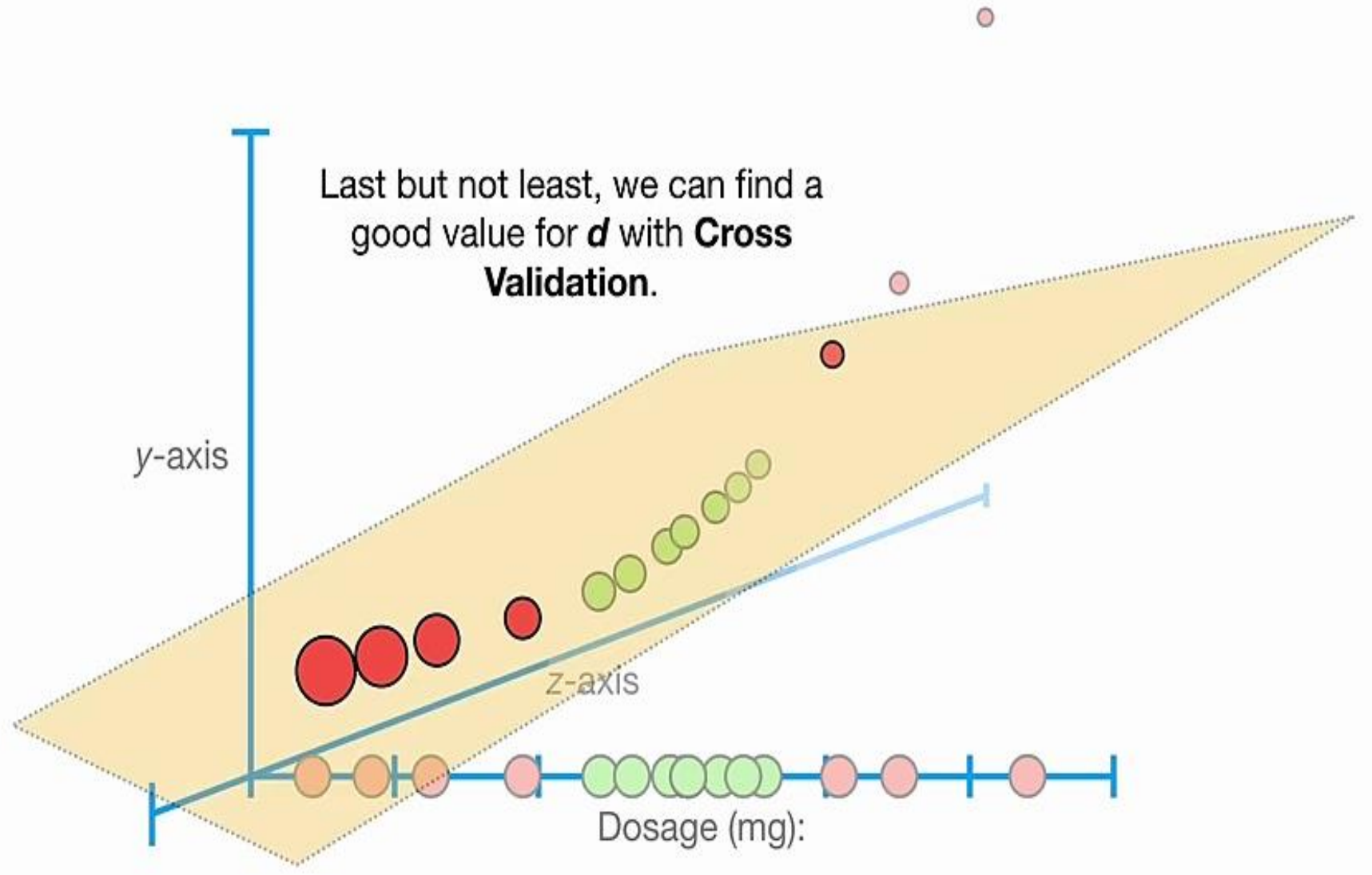
Another very commonly used **Kernel** is the **Radial Kernel**, also known as the **Radial Basis Function (RBF) Kernel**.
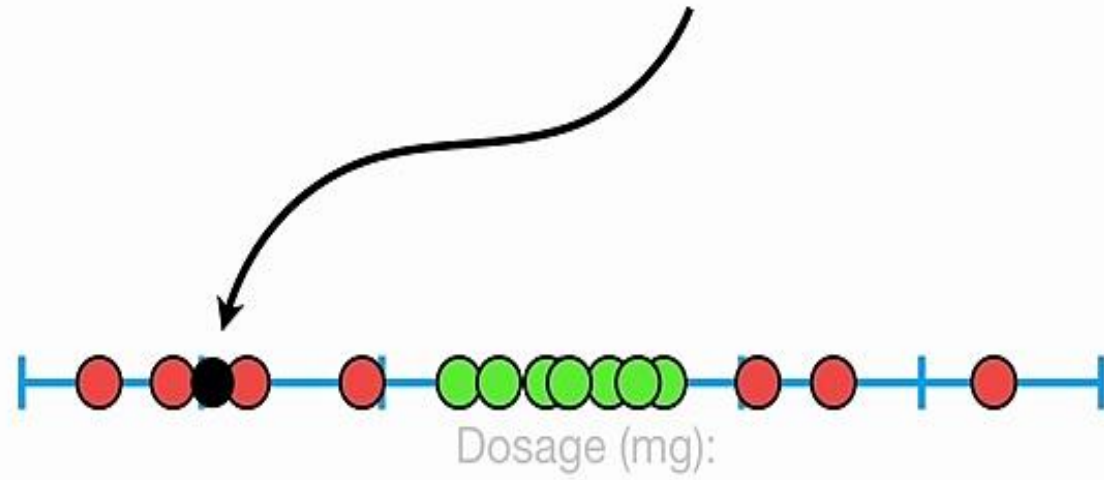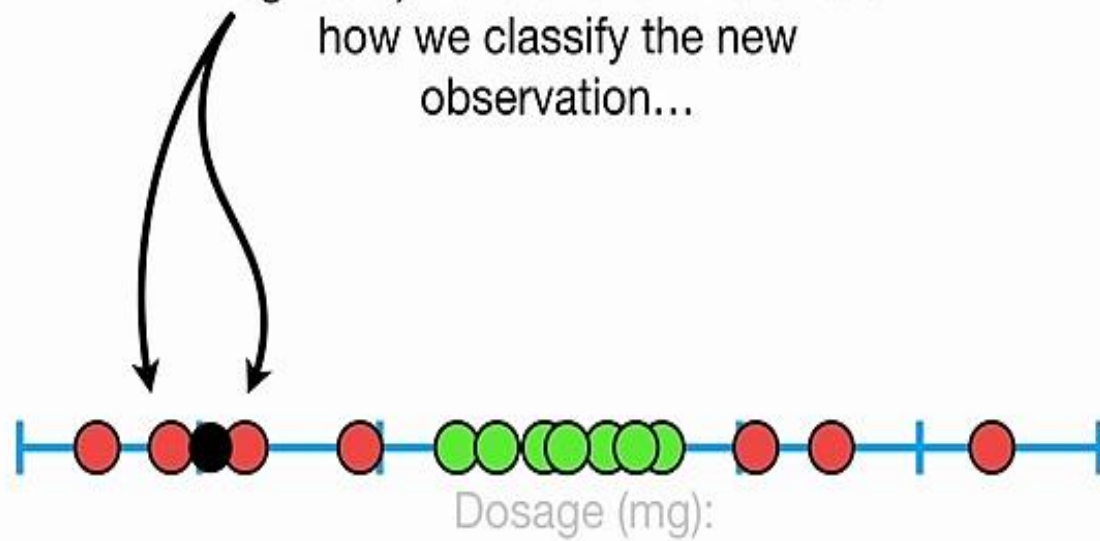
...and observations that are further away have relatively little influence on the classification.

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.

Dosage (mg):

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

Dosage (mg):
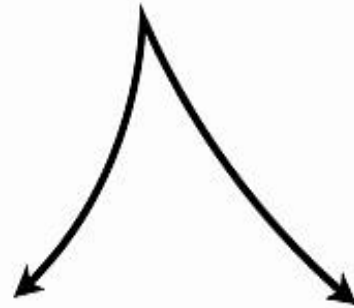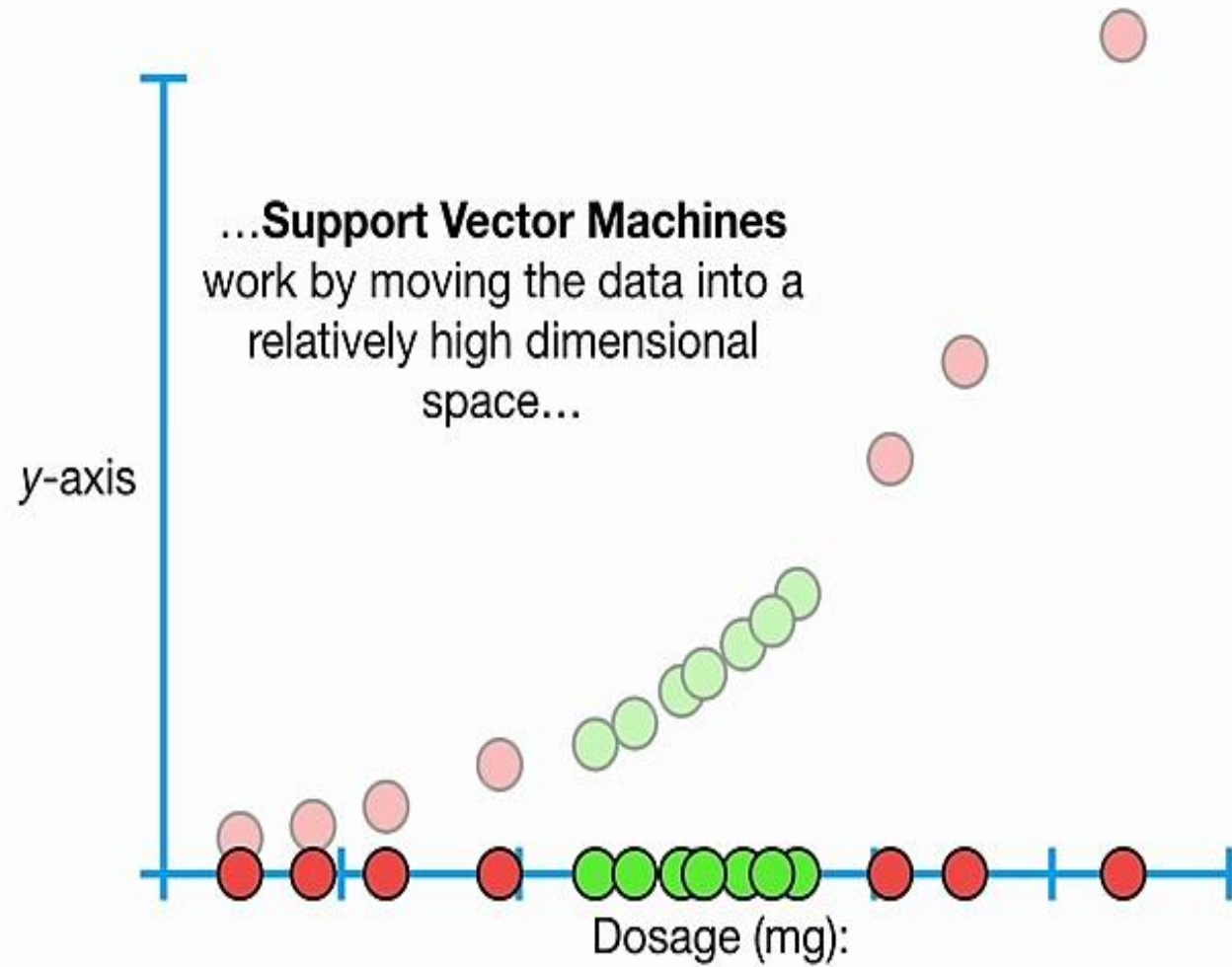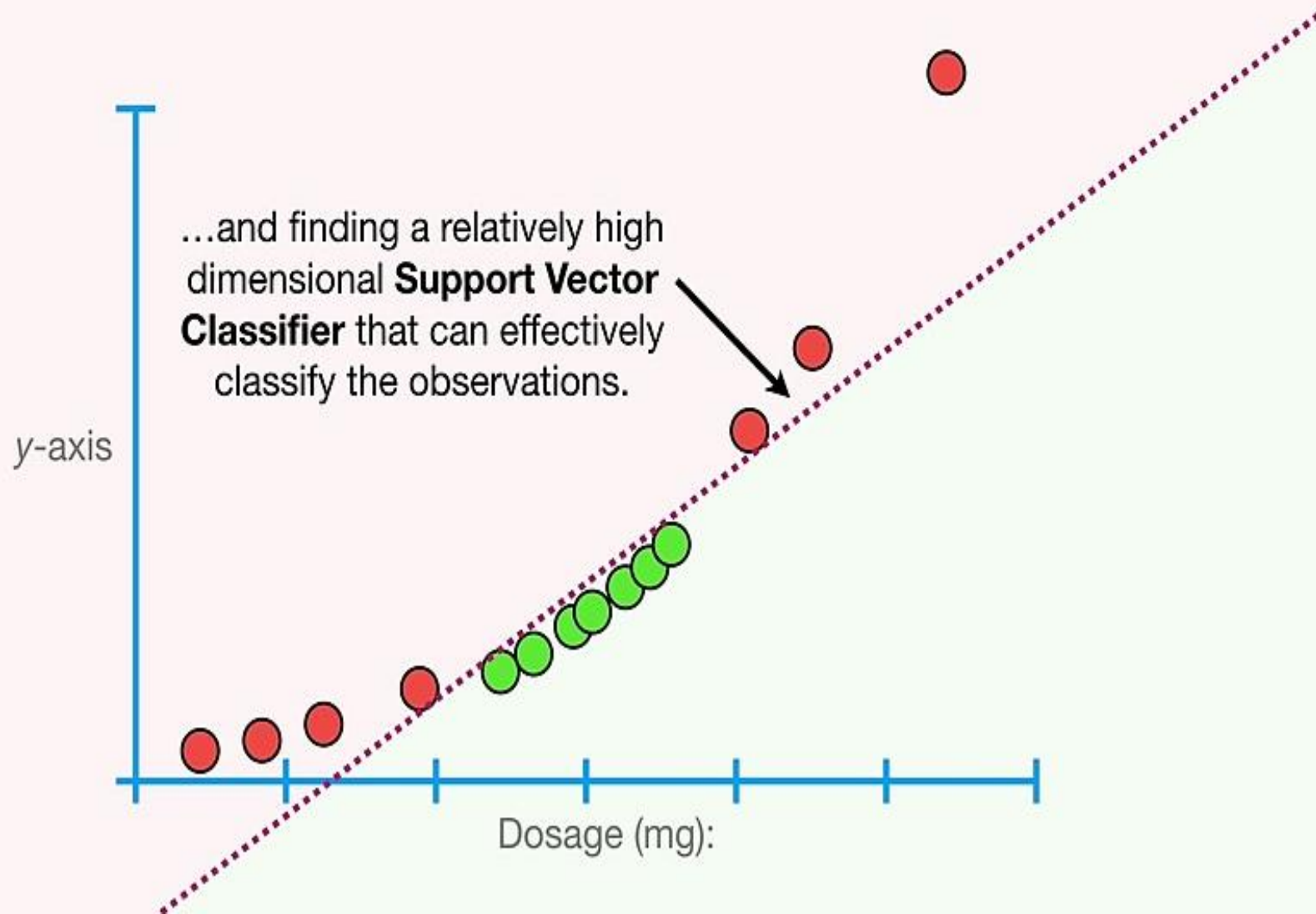
Non-linear SVM

Linearly separable
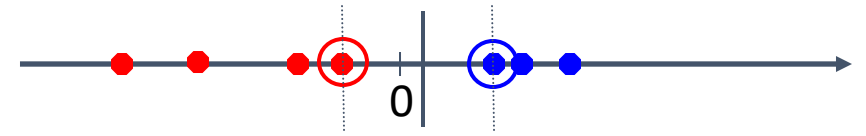
- What if the decision boundary is not linear?

Non-linearly separable

- How about... mapping data to a higher-dimensional space:
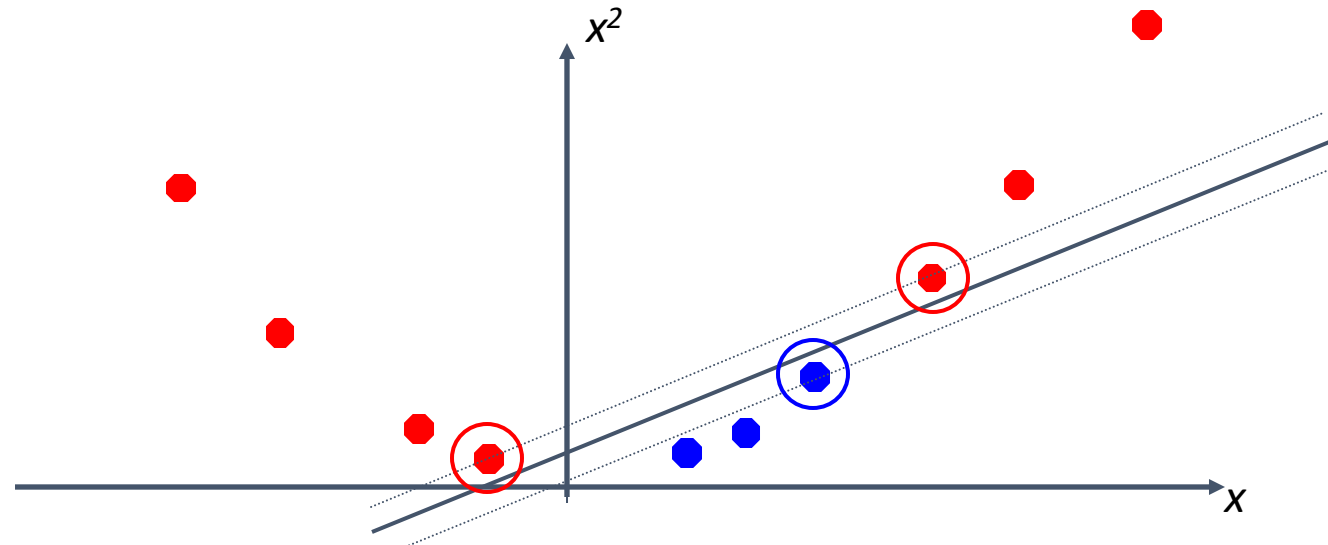
Non-linear SVMs: Feature Spaces

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

Φ: **x → ϕ(x)**

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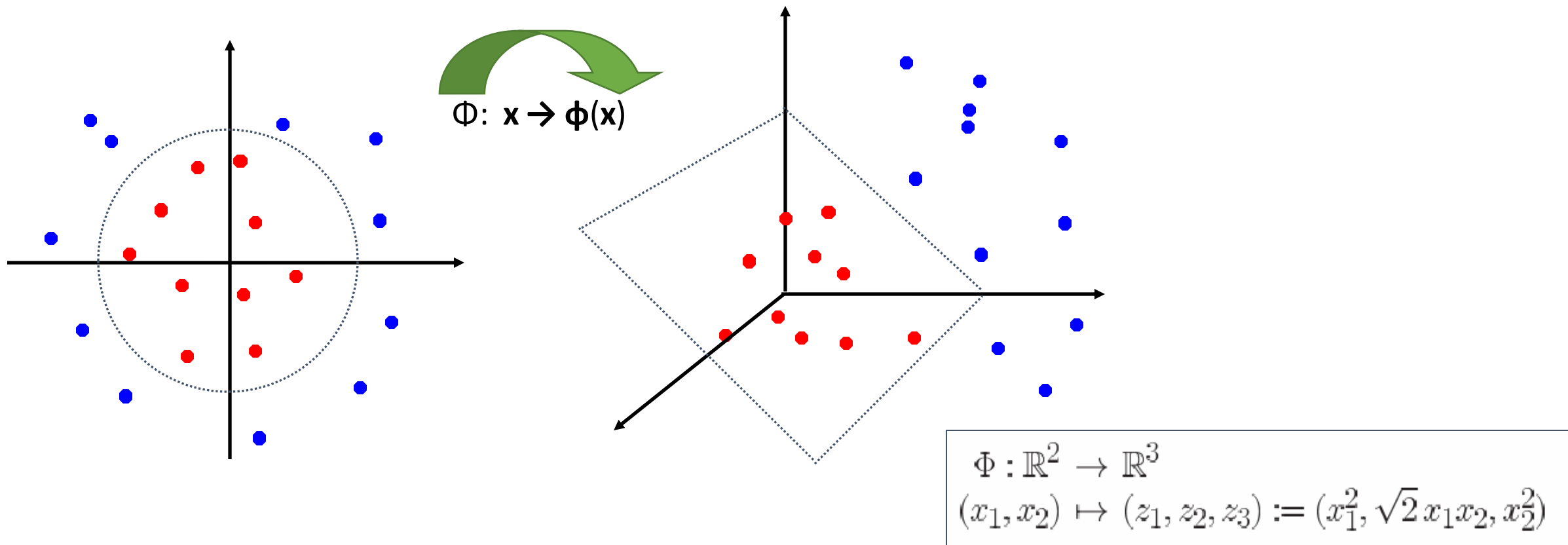
Non-linear SVM

- 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 $\vec{w} \bullet \Phi(\vec{x}) + b = 0$
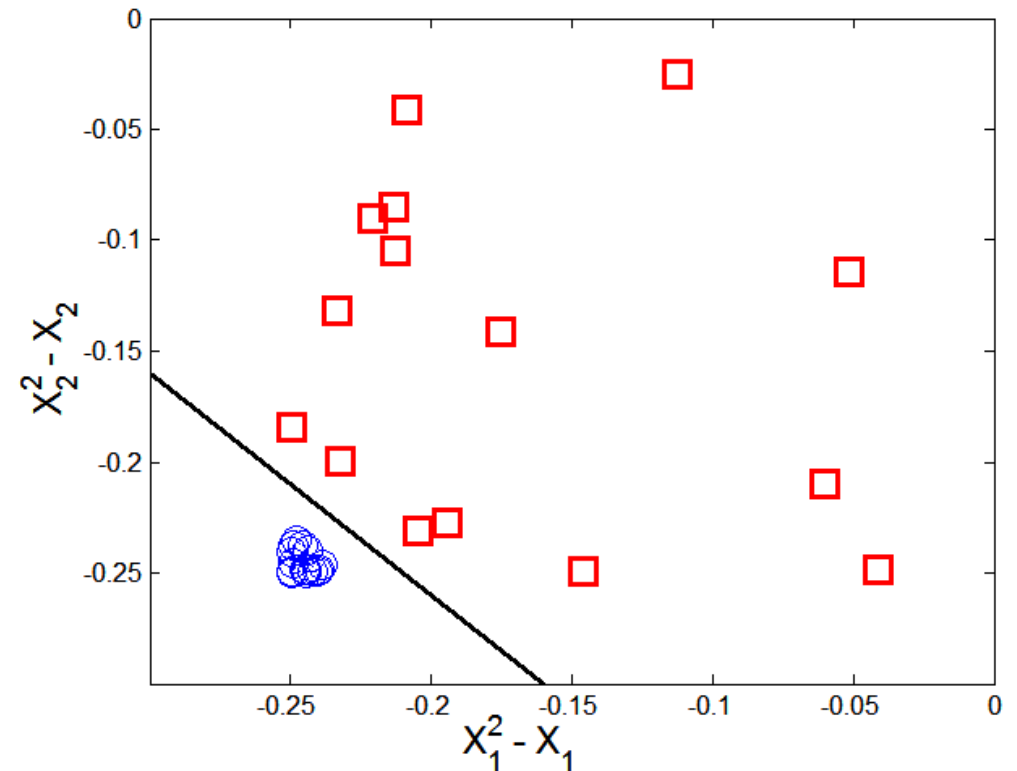
Learning a Nonlinear SVM

- Optimization problem

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

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

- 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
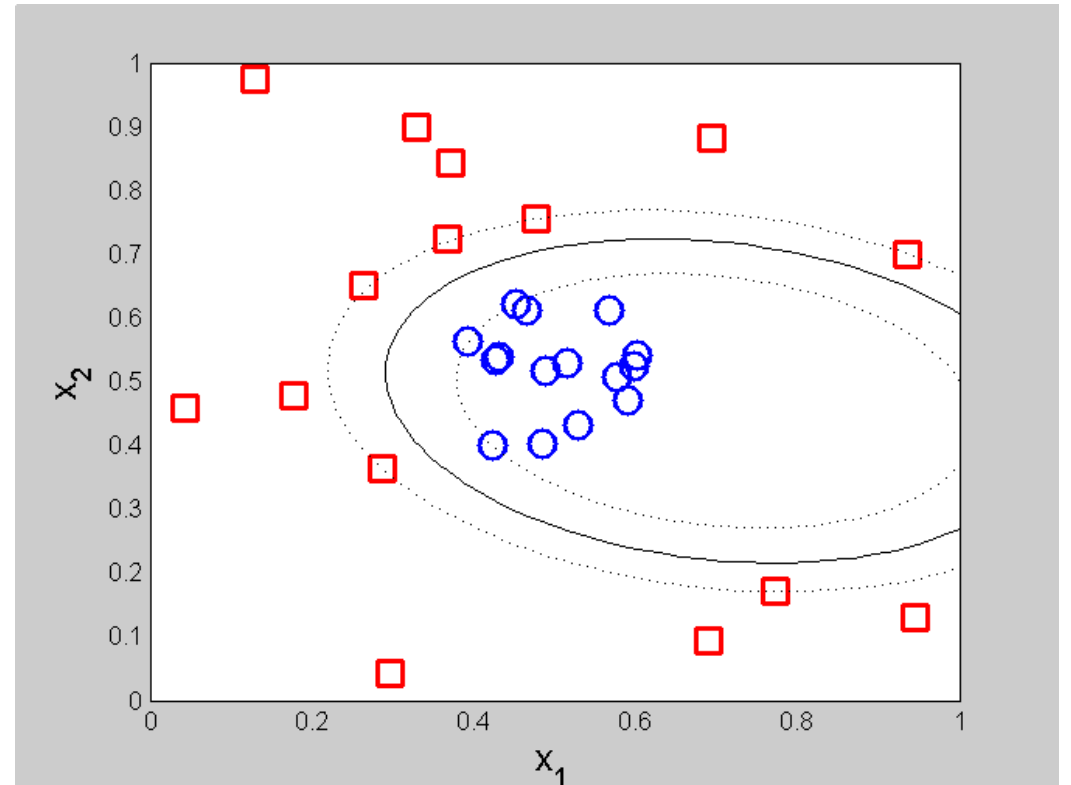
Examples of Kernel Functions

- 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 $\quad K(\mathbf{x}, \mathbf{y}) = \tanh(\kappa \mathbf{x}^T \mathbf{y} + \theta)$
  - 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 $K(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$
- If every datapoint is mapped into high-dimensional space via some transformation Φ: $\mathbf{x} \rightarrow \phi(\mathbf{x})$, the inner product becomes:

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

- 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 $K(\mathbf{x}_i,\mathbf{x}_j)=(1 + \mathbf{x}_i^T\mathbf{x}_j)^2$,

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

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

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

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

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

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 :

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

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

$\lambda = \alpha$

Example
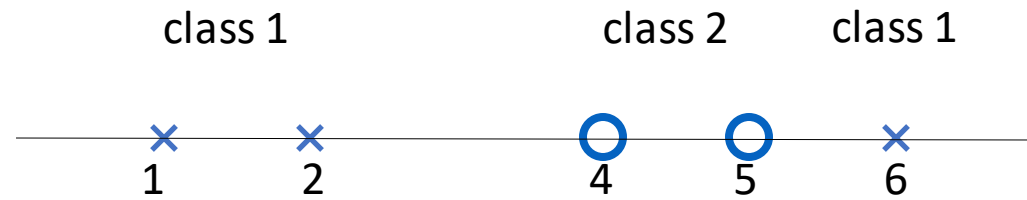
Example

- 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

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

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

Example

- We get
  - $\alpha_1=0$, $\alpha_2=2.5$, $\alpha_3=0$, $\alpha_4=7.333$, $\alpha_5=4.833$
  - Note that the constraints are indeed satisfied
  - The support vectors are $\{x_2=2, x_4=5, x_5=6\}$
- The discriminant function is

$$\overset{\alpha_5}{} \qquad \overset{y_5}{} \qquad \overset{K(z,x_5)}{}$$

$$f(z)$$
$$= 2.5(1)(2z+1)^2 + 7.333(-1)(5z+1)^2 + 4.833(1)(6z+1)^2 + b$$
$$= 0.6667z^2 - 5.333z + b$$

- *b* is recovered by solving f(2)=1 or by f(5)=-1 or by f(6)=1, as x₂ and x₅ lie on the line $\phi(\mathbf{w})^T\phi(\mathbf{x}) + b = 1$ and x₄ 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
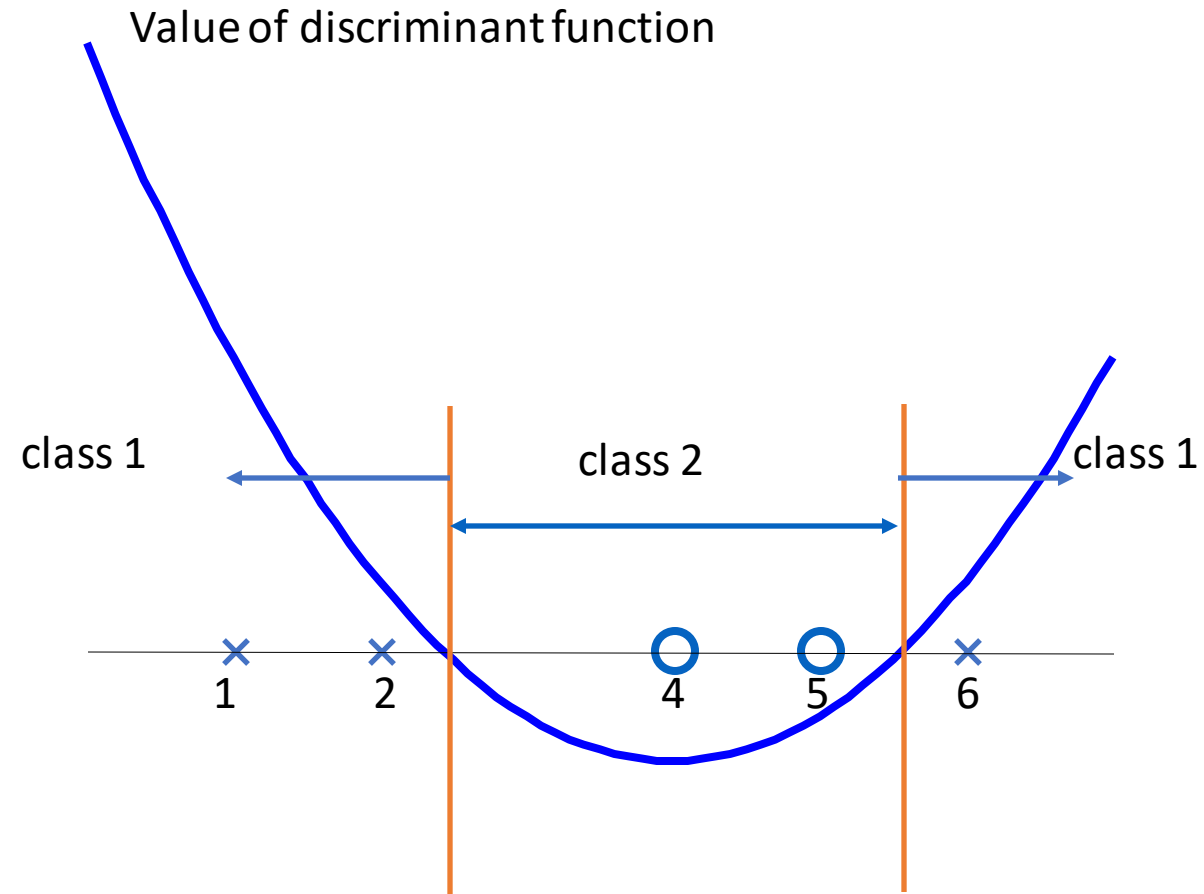
Support Vector Machine (SVM)

- 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.**
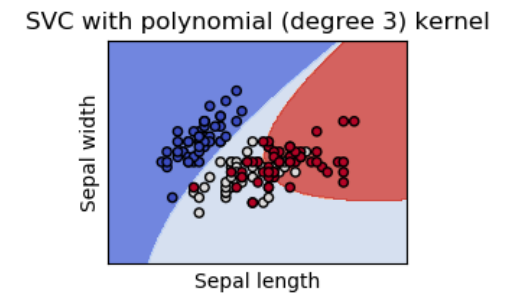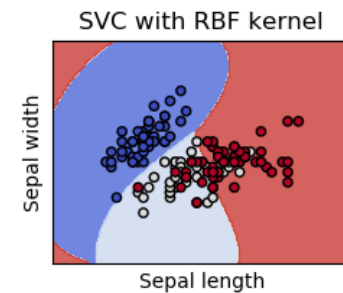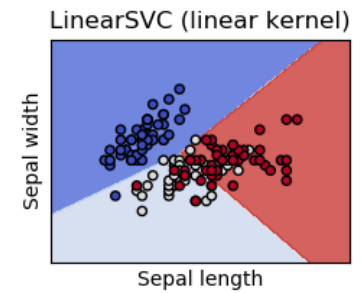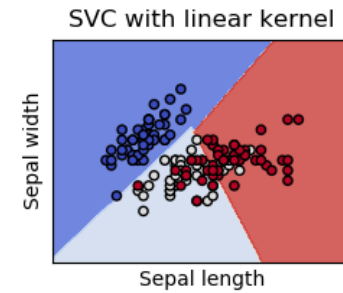
Characteristics of SVM

- 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.

Multiclass Classification

Multiclass Classification

- Combining binary classifiers
  - One-vs-all
  - All-vs-all

- Training a single classifier
  - Multiclass SVM
  - Constraint classification

Binary to Multiclass

- Can we use a binary classifier to construct a multiclass classifier?
  - Decompose the prediction into multiple binary decisions

- How to decompose?
  - One-vs-all
  - All-vs-all

One-vs-all Classification

- Assumption: Each class individually separable from **all** the others

- Learning: Given a dataset D = {<$\mathbf{x}_i$, $\mathbf{y}_i$>},
  Note: $\mathbf{x}_i$ 2 $<^n$, $\mathbf{y}_i$ 2 {1, 2, $\cdots$, K}
  - Decompose into K binary classification tasks
  - For class k, construct a binary classification task as:
    - Positive examples: Elements of D with label k
    - Negative examples: All other elements of D
  - Train K binary classifiers $\mathbf{w}_1$, $\mathbf{w}_2$, $\cdots$ $\mathbf{w}_K$ using any learning algorithm we have seen

- Prediction: "*Winner Takes All*"

$$\text{argmax}_i \ \mathbf{w}_i^\top \mathbf{x}$$

Question: What is the dimensionality of each $\mathbf{w}_i$?

Visualizing One-vs-All

From the full dataset, construct three binary classifiers, one for each class

$\mathbf{w_{blue}}^\top\mathbf{x} > 0$
for **blue inputs**

$\mathbf{w_{red}}^\top\mathbf{x} > 0$
for **red inputs**

$\mathbf{w_{green}}^\top\mathbf{x} > 0$
for **green inputs**

Notation: Score for blue label

Winner Take All will predict the right answer. Only the correct label will have a positive score
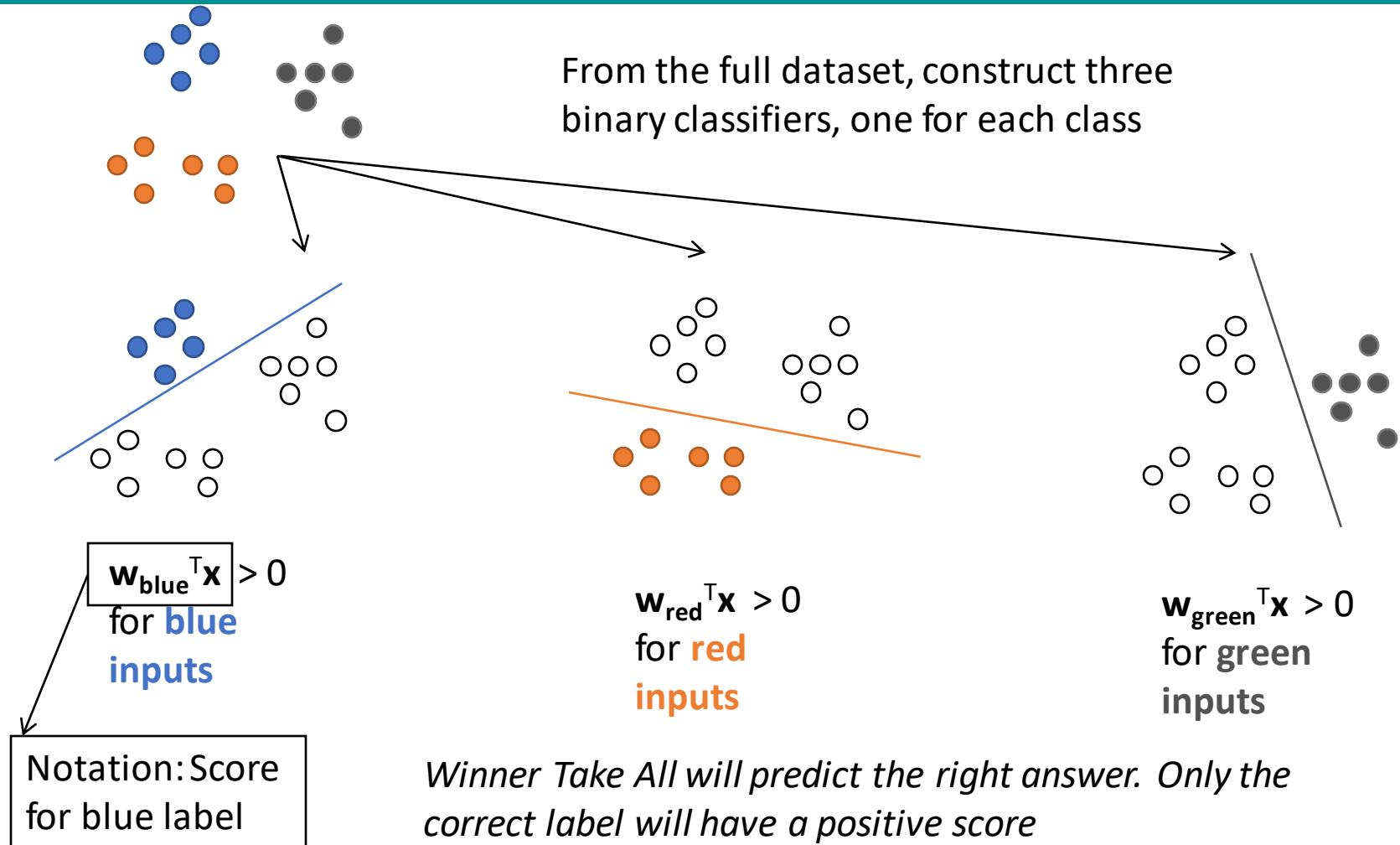
One-vs-All May not Always Work
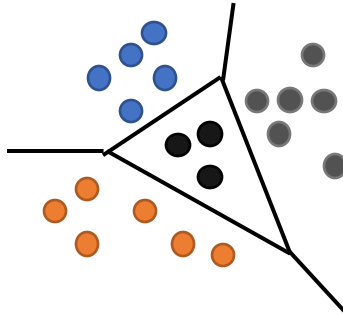
Black points are not separable with a single binary classifier

The decomposition will not work for these cases!
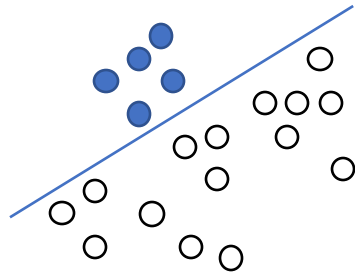
$\mathbf{w_{blue}}^\mathsf{T}\mathbf{x} > 0$
for **blue**
inputs
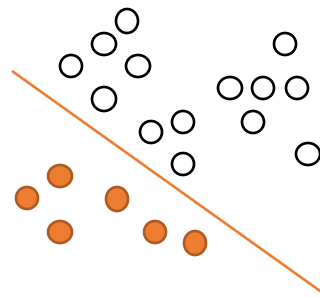
$\mathbf{w_{red}}^\mathsf{T}\mathbf{x} > 0$
for **red**
inputs
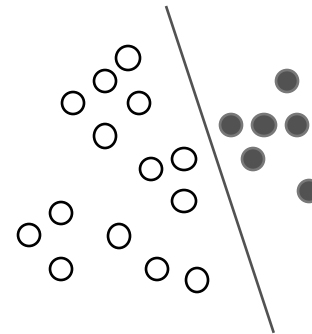
$\mathbf{w_{green}}^\mathsf{T}\mathbf{x} > 0$
for **green**
inputs

???
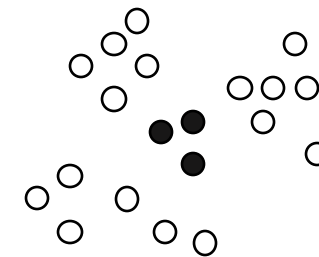
One-vs-All Classification: Summary

- Easy to learn
  - Use any binary classifier learning algorithm
- Problems
  - No theoretical justification
  - Calibration issues
    - We are comparing scores produced by K classifiers trained independently. No reason for the scores to be in the same numerical range!
  - Might not always work
    - Yet, works fairly well in many cases, especially if the underlying binary classifiers are tuned, regularized

All-vs-All Classification

- Assumption: *Every* pair of classes is separable
- Learning: Given a dataset D = {<x_i, y_i>},
  Note: $x_i \in \Re^n$, $y_i \in \{1, 2, \cdots, K\}$
  - For every pair of labels (j, k), create a binary classifier with:
    - Positive examples: All examples with label j
    - Negative examples: All examples with label k

  - Train $\begin{pmatrix} K \\ 2 \end{pmatrix} = \dfrac{K(K-1)}{2}$ classifiers in all

- Prediction: More complex, each label get K-1 votes
  - How to combine the votes? Many methods
    - Majority: Pick the label with maximum votes
    - Organize a tournament between the labels

All-vs-All Classification
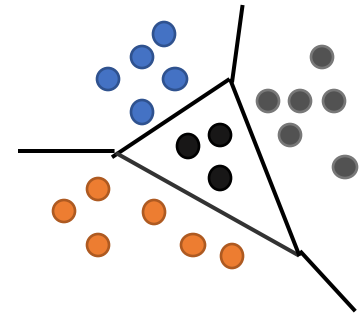
- Every pair of labels is linearly separable here
  - When a pair of labels is considered, all others are ignored
- Problems
  1. $O(K^2)$ weight vectors to train and store
  2. Size of training set for a pair of labels could be very small, leading to overfitting
  3. Prediction is often ad-hoc and might be unstable

     Eg: What if two classes get the same number of votes? For a tournament, what is the sequence in which the labels compete?

Training a Single Classifier: Motivation

- Decomposition methods
  - Do not account for how the final predictor will be used
  - Do not optimize any global measure of correctness

- Goal: To train a multiclass classifier that is "global"

Multiclass Margin

Defined as the score difference between the highest scoring label and the second one
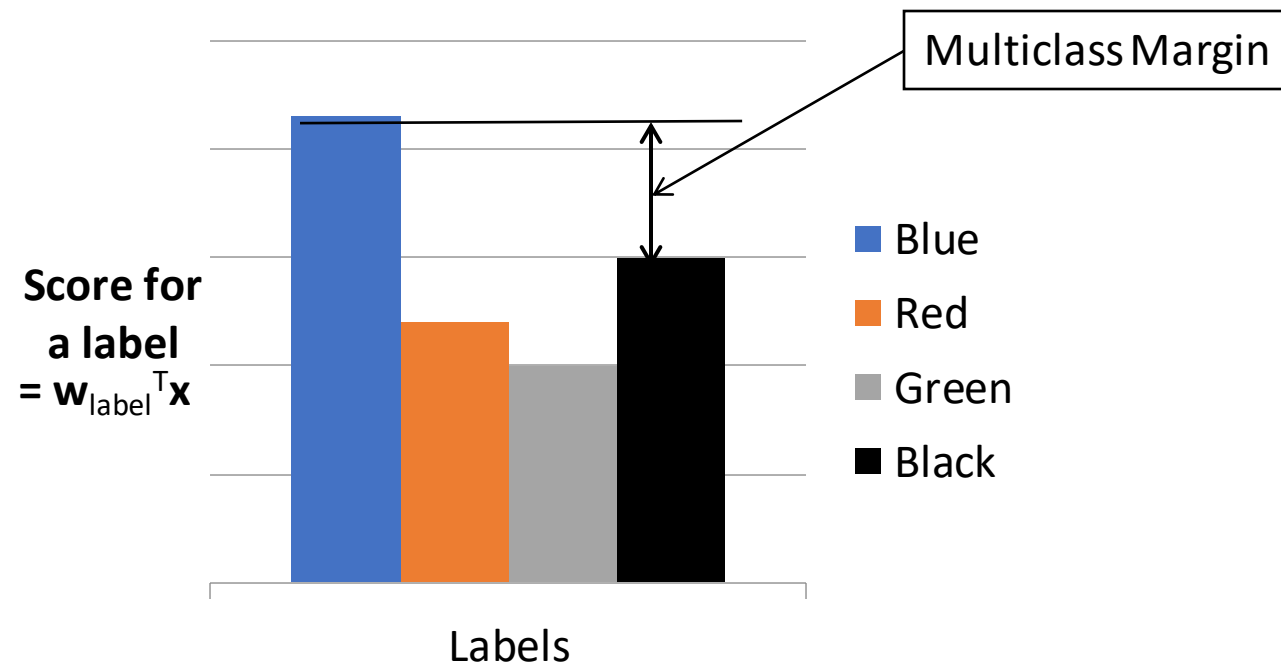
Multiclass SVM (Intuition)

- Recall: Binary SVM
  - Maximize margin
  - Equivalently,
    Minimize norm of weights such that the closest points to the hyperplane have a score §1

- Multiclass SVM
  - Each label has a different weight vector (like one-vs-all)
  - Maximize multiclass margin
  - Equivalently,
    Minimize total norm of the weights such that the true label is scored at least 1 more than the second best one

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.