

# Support Vector Machines

# Support Vector Machines

- Support Vector Machines are one of the more complex algorithms we will learn, but it all begins with a simple premise:
  - Does a hyperplane exist that can effectively separate classes?
- To answer this question, we first need to go through the history and development of Support Vector Machines!

# Support Vector Machines

- Section Overview
  - History
  - Intuition and Theory for SVM
  - SVM Classification Example
  - SVM Regression Example
  - SVM Project Exercise and Solutions

# Support Vector Machines

- Relevant Reading in ISLR
  - Chapter 9 covers Support Vector Machine Classification
  - Wikipedia has a section on Support Vector Machine Regression.

# Support Vector Machines

Theory and Intuition - History

# Support Vector Machines

- Support Vector Machines will be one of the most recently developed machine learning algorithms we will learn about!
- Let's briefly review the recent history behind the development of Support Vector Machines.

# Support Vector Machines

- Quick Note:
  - Spelling of some names and article titles may be slightly altered due to various translation options from Russian to English (e.g. Alexey vs. Alexei).

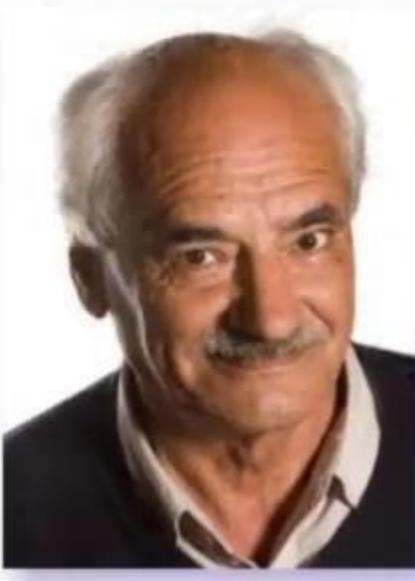
# Support Vector Machines

- 1960s: Vladimir Vapnik obtained his Ph.D in statistics at the Institute of Control Sciences in Moscow.
- Worked at the Institute from 1961-1990.



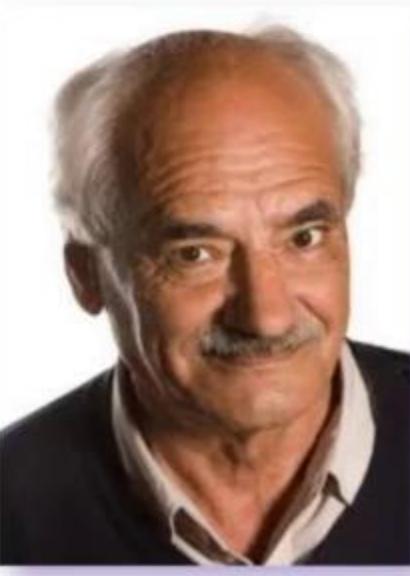
# Support Vector Machines

- 1960s: Alexey Chervonenkis and Vladimir Vapnik begin the development of Support Vector Machines.



# Support Vector Machines

- 1963: Published “Generalized Portrait Algorithm” for image analysis by computers.

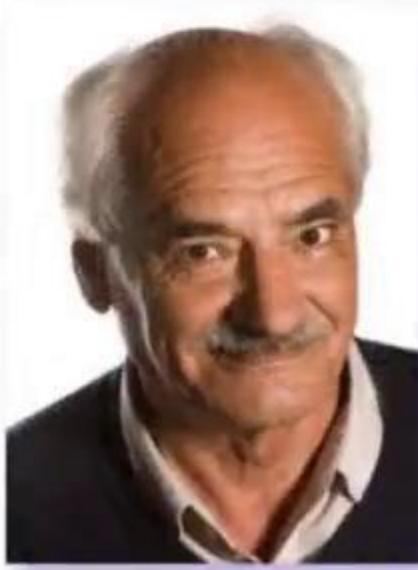


# Support Vector Machines

- 1964: Aizerman, Braverman, and Rozonoer, publish “*Theoretical Foundations of the Potential Function Method*” in *Pattern Recognition Learning*.
- Introduces the geometrical interpretation of the kernels as inner products in a feature space.

# Support Vector Machines

- 1974: Vapnik and Chervonenkis continue work, publishing “**Theory of Pattern Recognition**”.



# Support Vector Machines

- 1960s-1990s: Vapnik continues to work on further developing Support Vector Machines.



# Support Vector Machines

- 1992: Bernhard Boser, Isabelle Guyon and Vladimir Vapnik suggested a way to create nonlinear classifiers by applying the **kernel trick** to maximum-margin hyperplanes.



# Support Vector Machines

- 1995: Corinna Cortes and Vladimir Vapnik publish the SVM incarnation utilizing a **soft margin**.



# Support Vector Machines

- 1996: Vladimir Vapnik, Harris Drucker, Christopher Burges, Linda Kaufman and Alexander Smola publish “Support Vector Regression Machines”, expanding SVM beyond classification tasks.

# Support Vector Machines

- It took over 30 years of work to get to the version of Support Vector Machines we will be using!
- Work continues today with improvements in implementation and performance, as well as theoretical advancements.

# Support Vector Machines

- As we begin to learn about the intuition and theory behind Support Vector Machines, keep in mind this steady progress.
- We will see it advance clearly as we move from Maximum Margin Classifiers, to Support Vector Classifiers, and finally Support Vector Machines.



# Support Vector Machines

Theory and Intuition - Hyperplanes and Margins

# Support Vector Machines

- We will slowly build up to SVMs:
  - Maximum Margin Classifier
  - Support Vector Classifier
  - Support Vector Machines
- Let's begin by understanding what a hyperplane is...

# Support Vector Machines

- In an  $N$ -dimensional space, a hyperplane is a flat affine subspace of dimension  $N - 1$ .
- For example:
  - 1-D Hyperplane is a single point
  - 2-D Hyperplane is a line
  - 3-D Hyperplane is flat plane

# Support Vector Machines

- 1-D Hyperplane is a single point



# Support Vector Machines

- 1-D Hyperplane is a single point



# Support Vector Machines

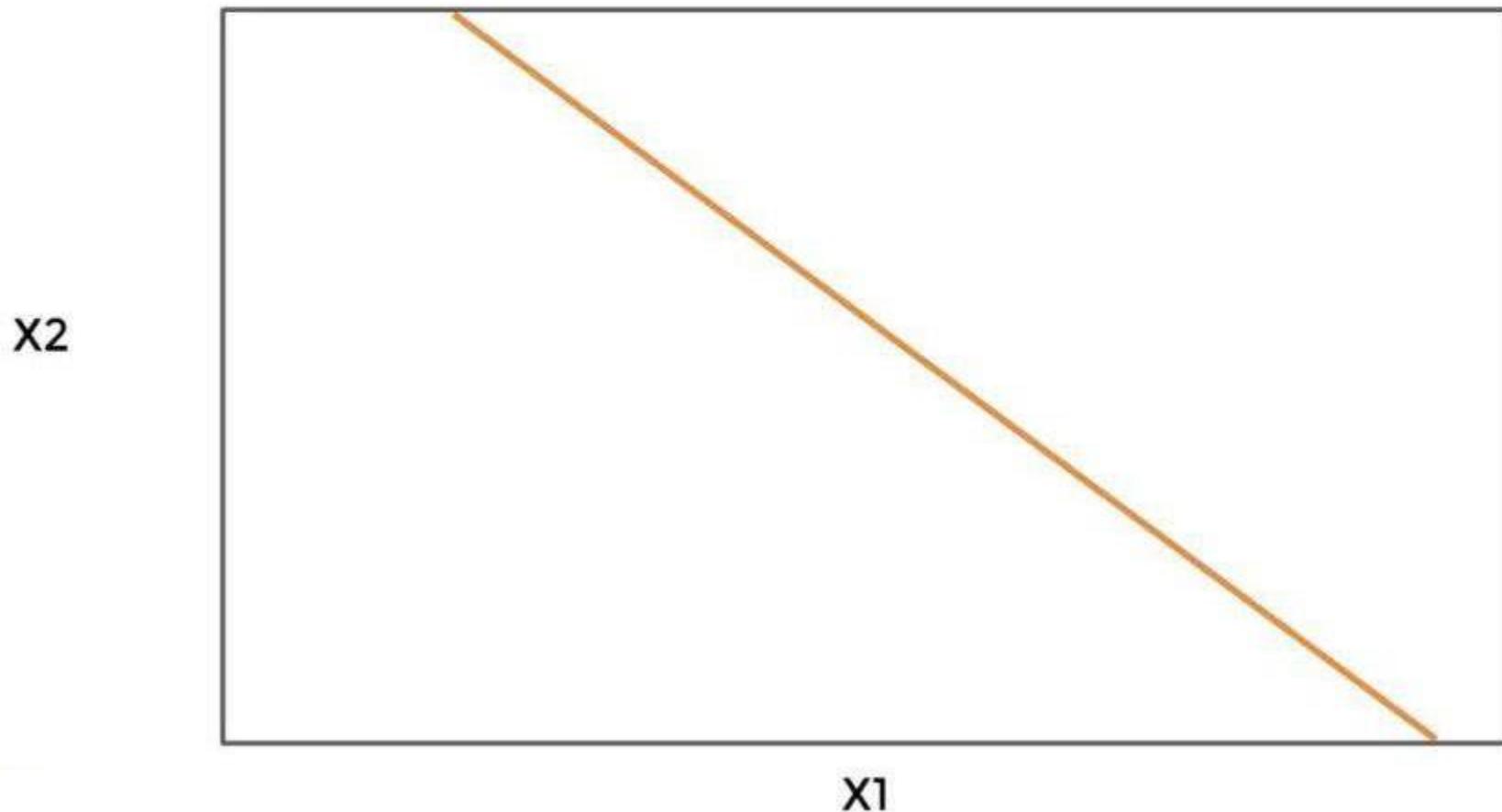
- 2-D Hyperplane is a line

$x_2$

$x_1$

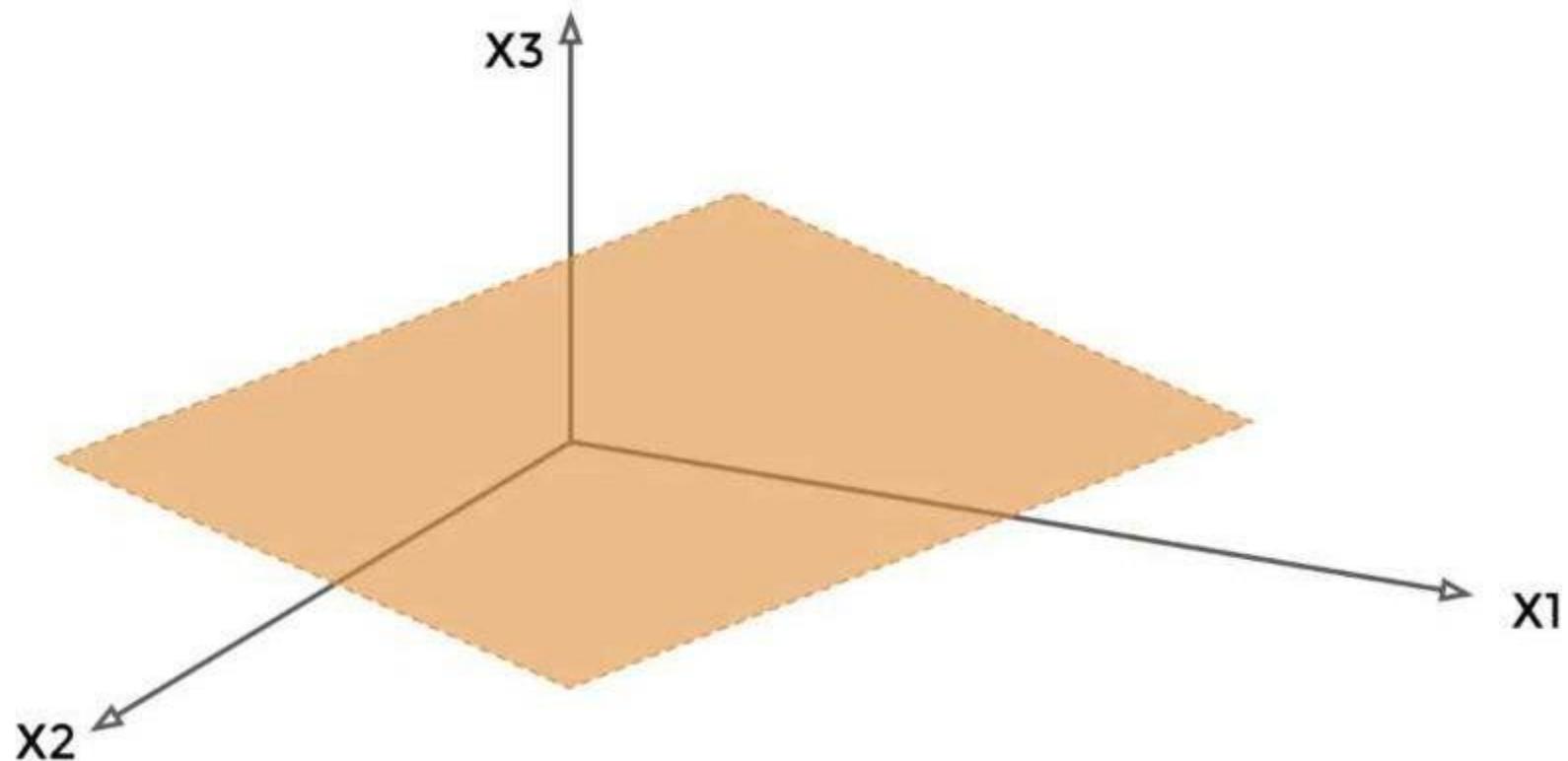
# Support Vector Machines

- 2-D Hyperplane is a line



# Support Vector Machines

- 3-D Hyperplane is a flat plane



# Support Vector Machines

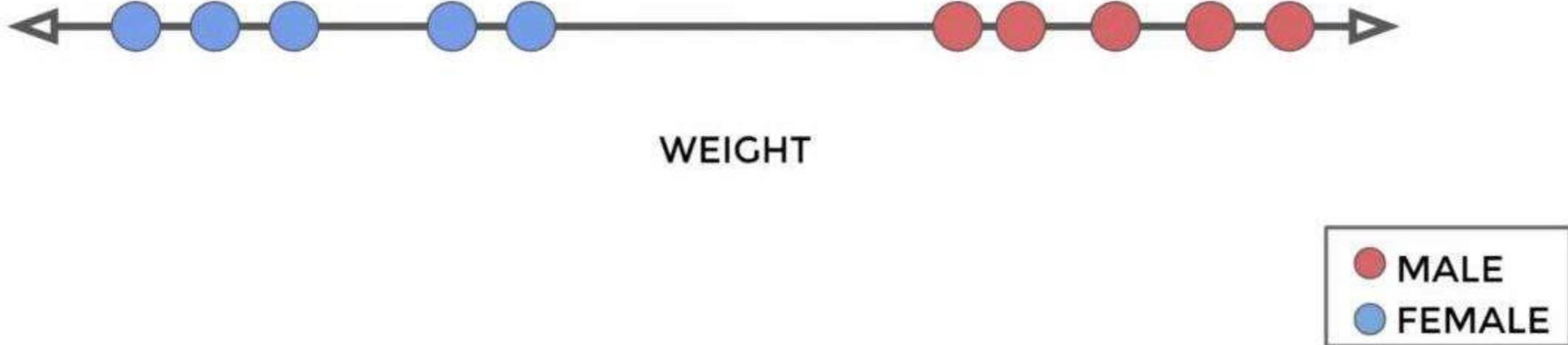
- The main idea behind SVM is that we can use Hyperplanes to create a separation between classes.
- Then new points will fall on one side of this separating hyperplane, which we can then use to assign a class.

# Support Vector Machines

- Imagine a data set with one feature and one binary target label.
- For example:
  - A weight feature for baby chicks
  - Classified by Male or Female
- What would this look like visualized?

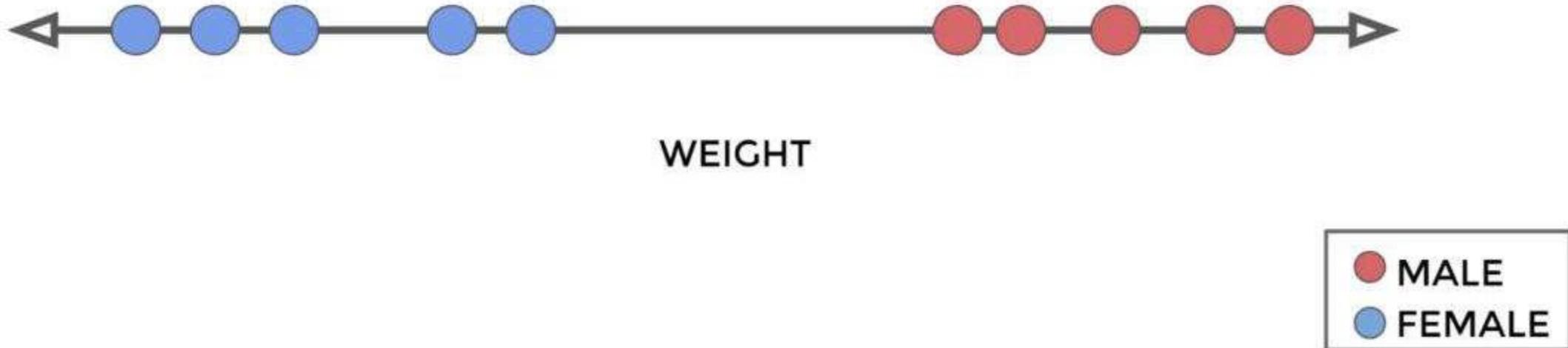
# Support Vector Machines

- Place points along feature.



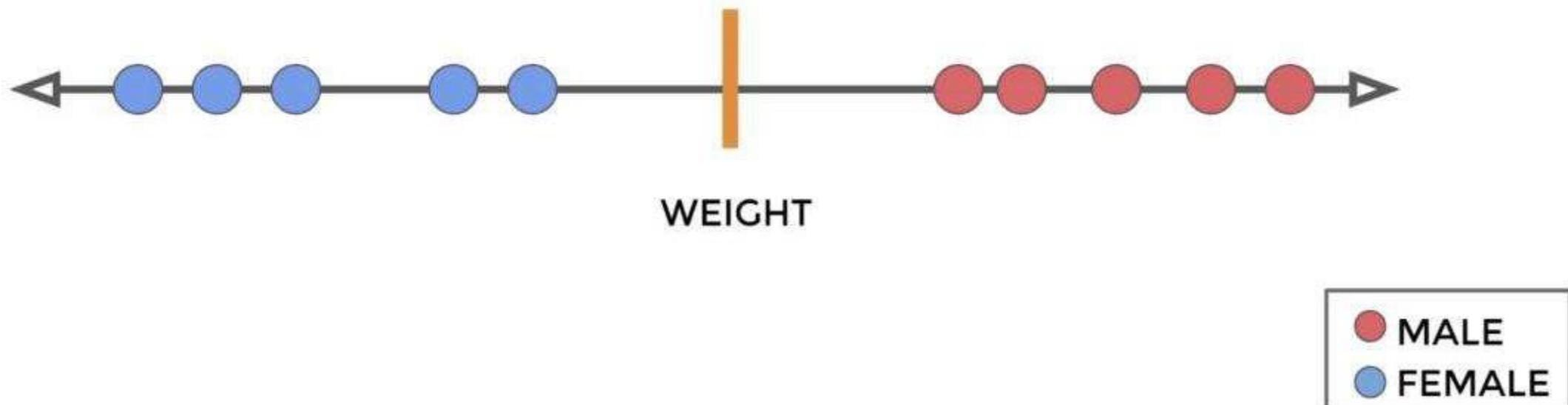
# Support Vector Machines

- Notice in this case, classes are perfectly separable. This is unlikely in real world datasets.



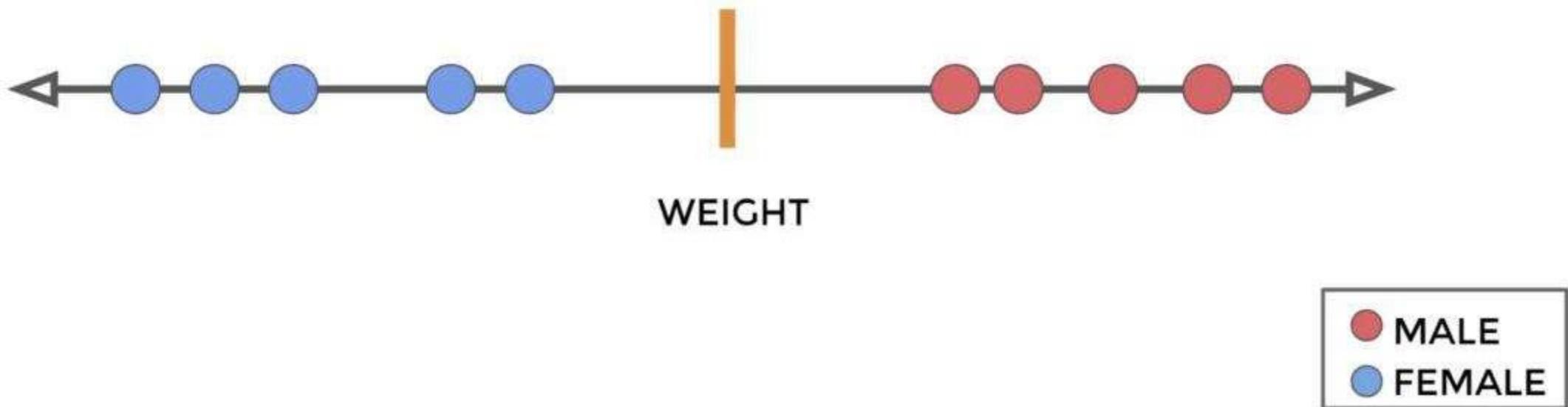
# Support Vector Machines

- Idea behind SVM is to create a **separating hyperplane** between the classes.



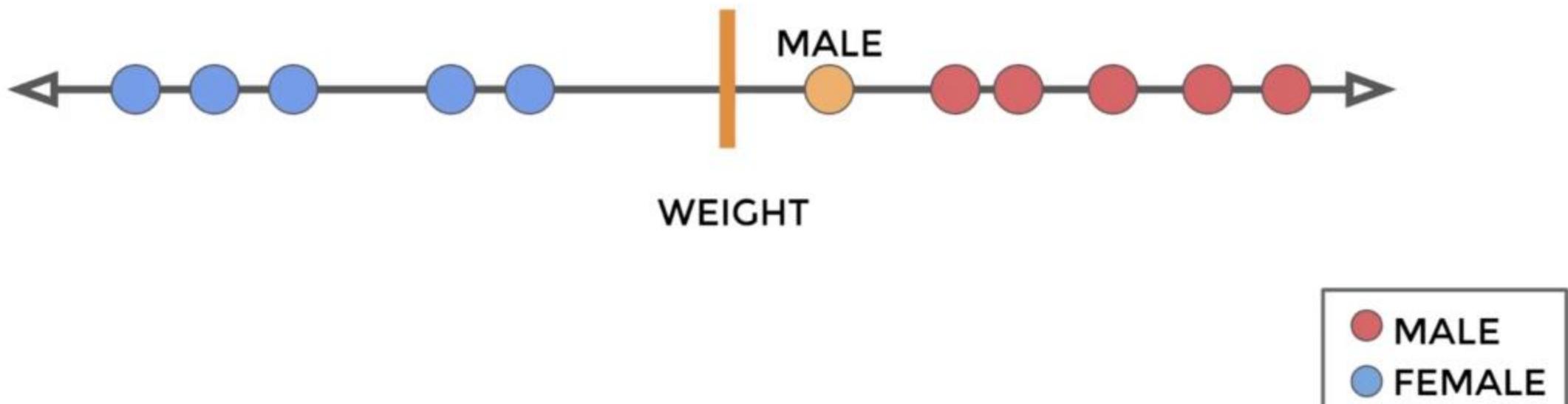
# Support Vector Machines

- A new point would be classified based on what side of the point they land on.



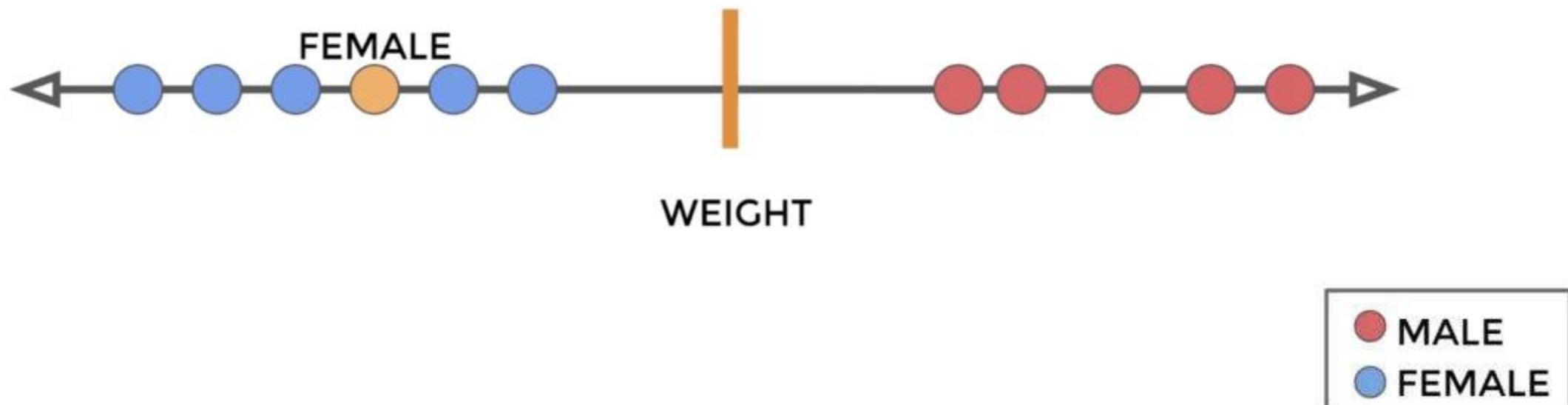
# Support Vector Machines

- A new point would be classified based on what side of the point they land on.



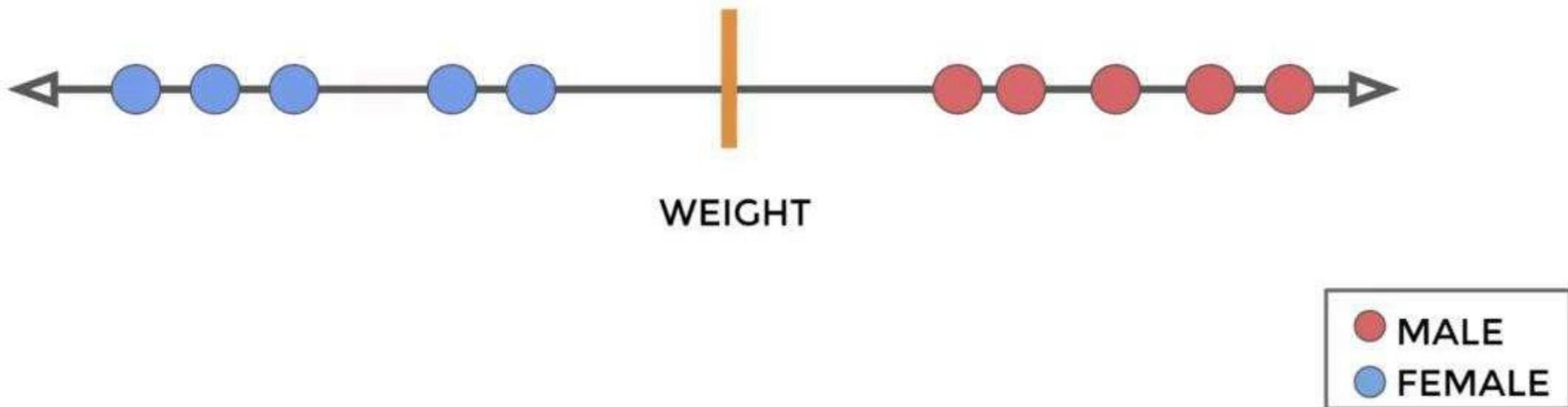
# Support Vector Machines

- A new point would be classified based on what side of the point they land on.



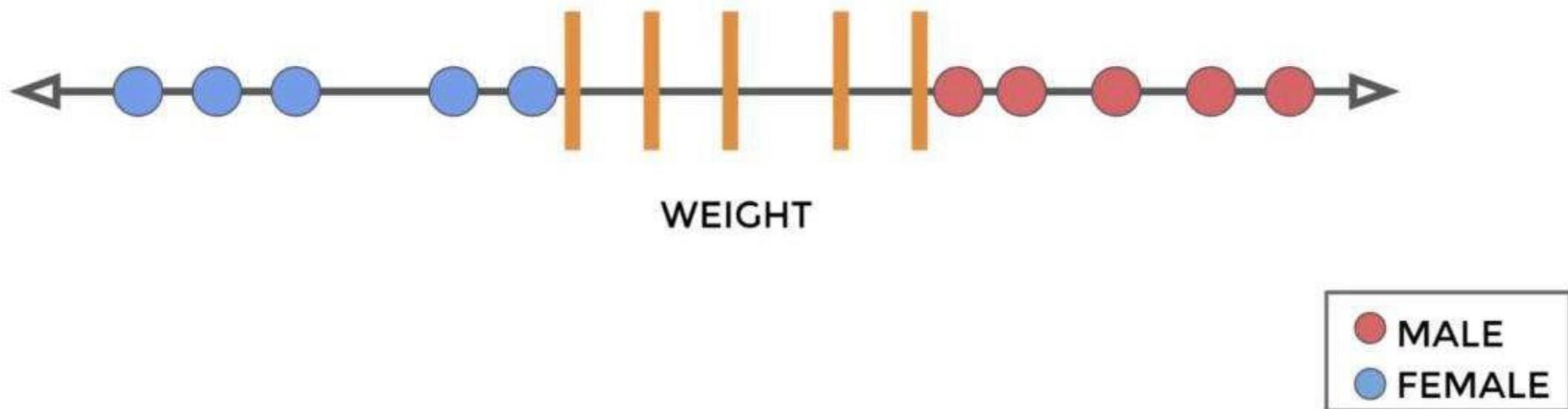
# Support Vector Machines

- How do we choose where to put this separating hyperplane?



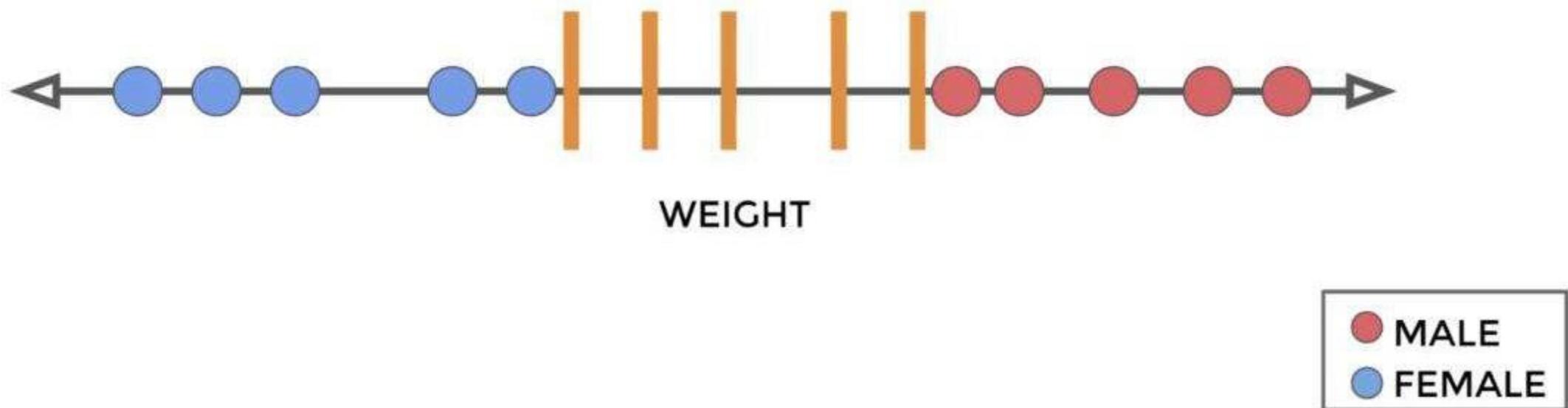
# Support Vector Machines

- Note there are many options that perfectly separate out these classes:



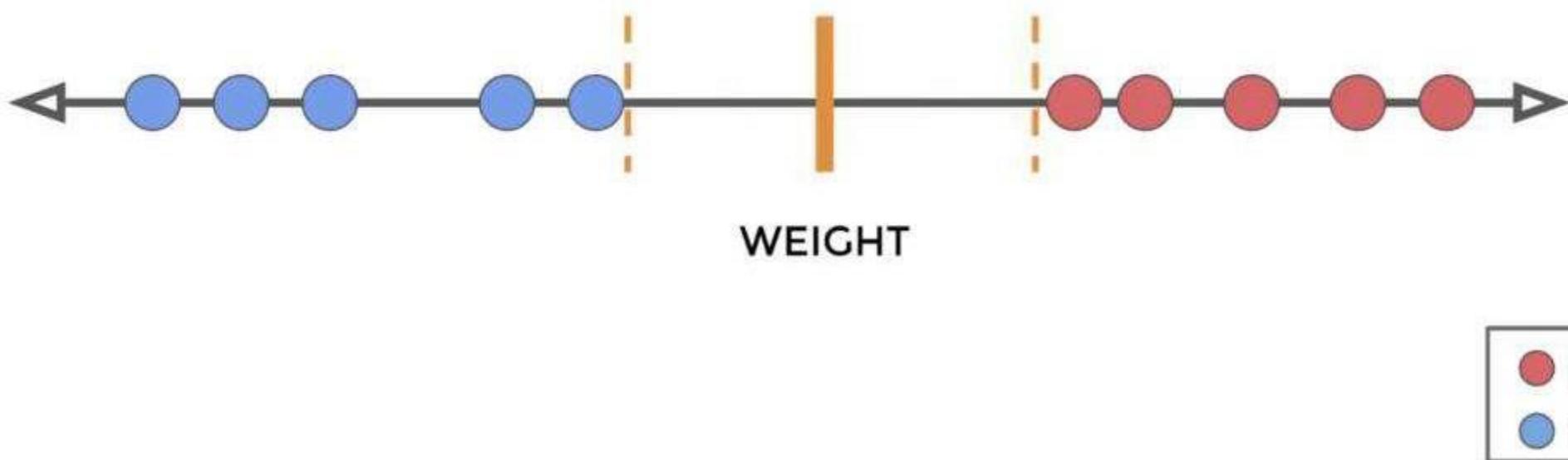
# Support Vector Machines

- Which of these is the “best” separator between the classes?



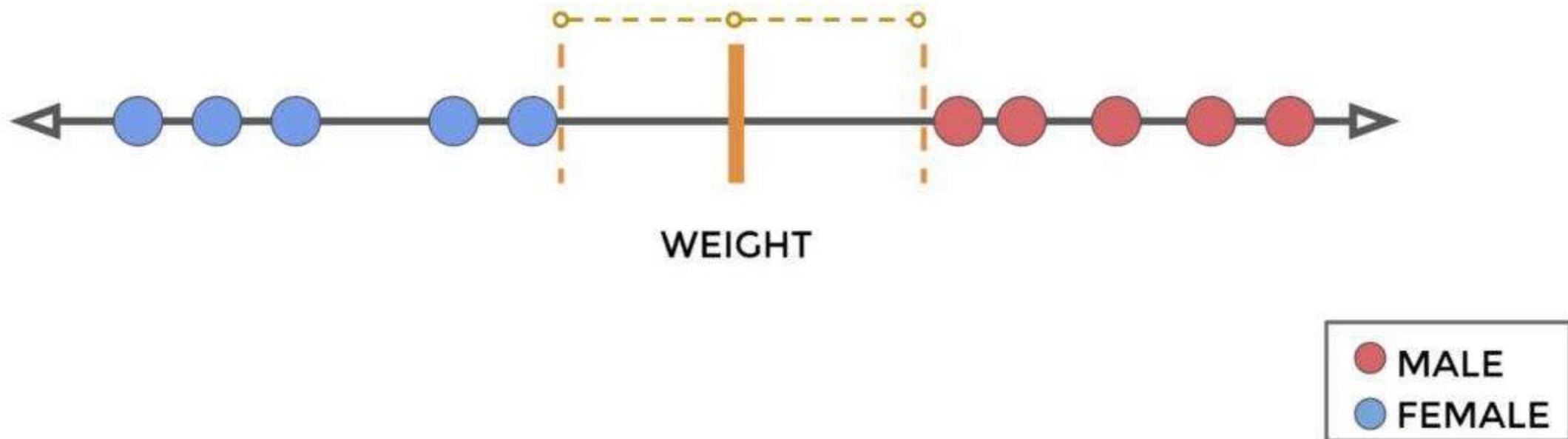
# Support Vector Machines

- We could use the separator that maximizes the margins between the classes.



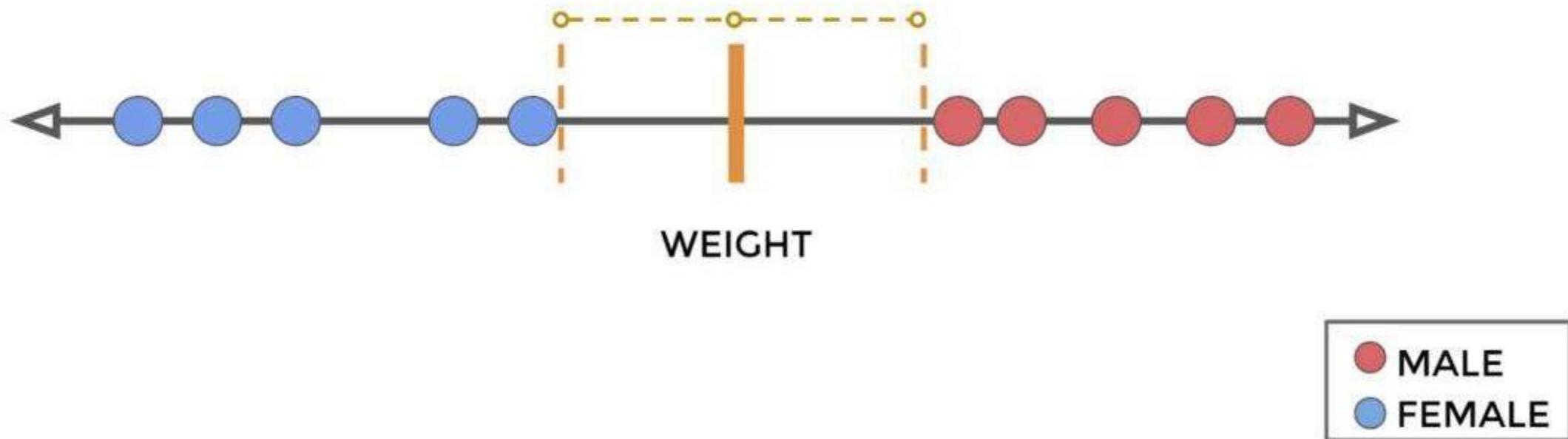
# Support Vector Machines

- This is known as a **Maximal Margin Classifier**.



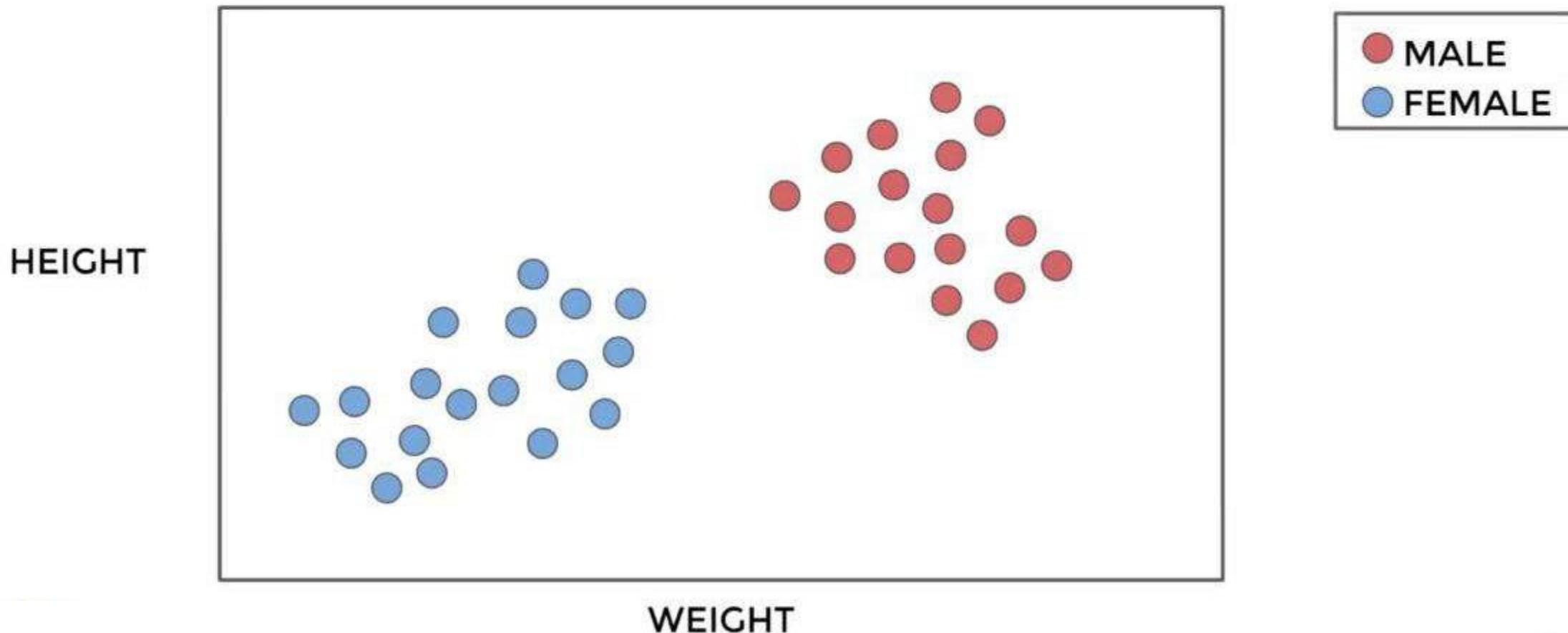
# Support Vector Machines

- This same idea of maximum margins applies to N-dimensions.



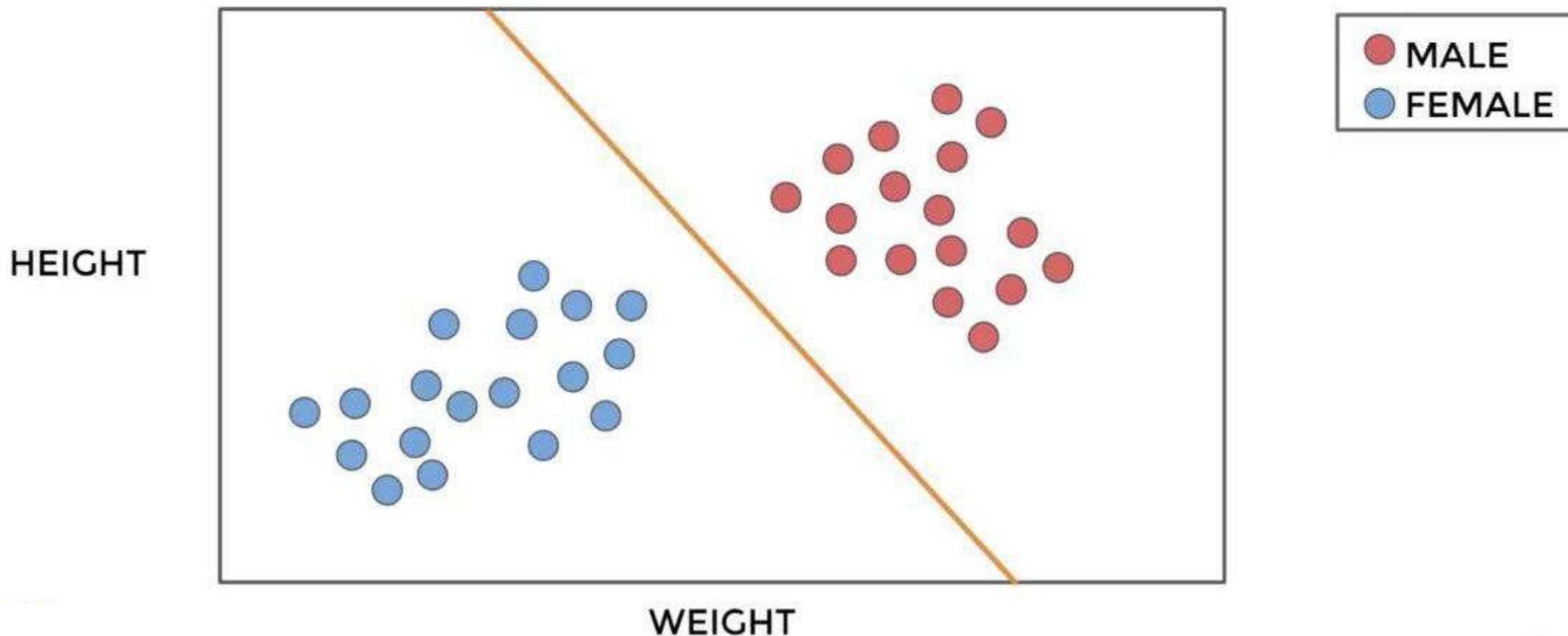
# Support Vector Machines

- Imagine a 2 dimensional feature space:



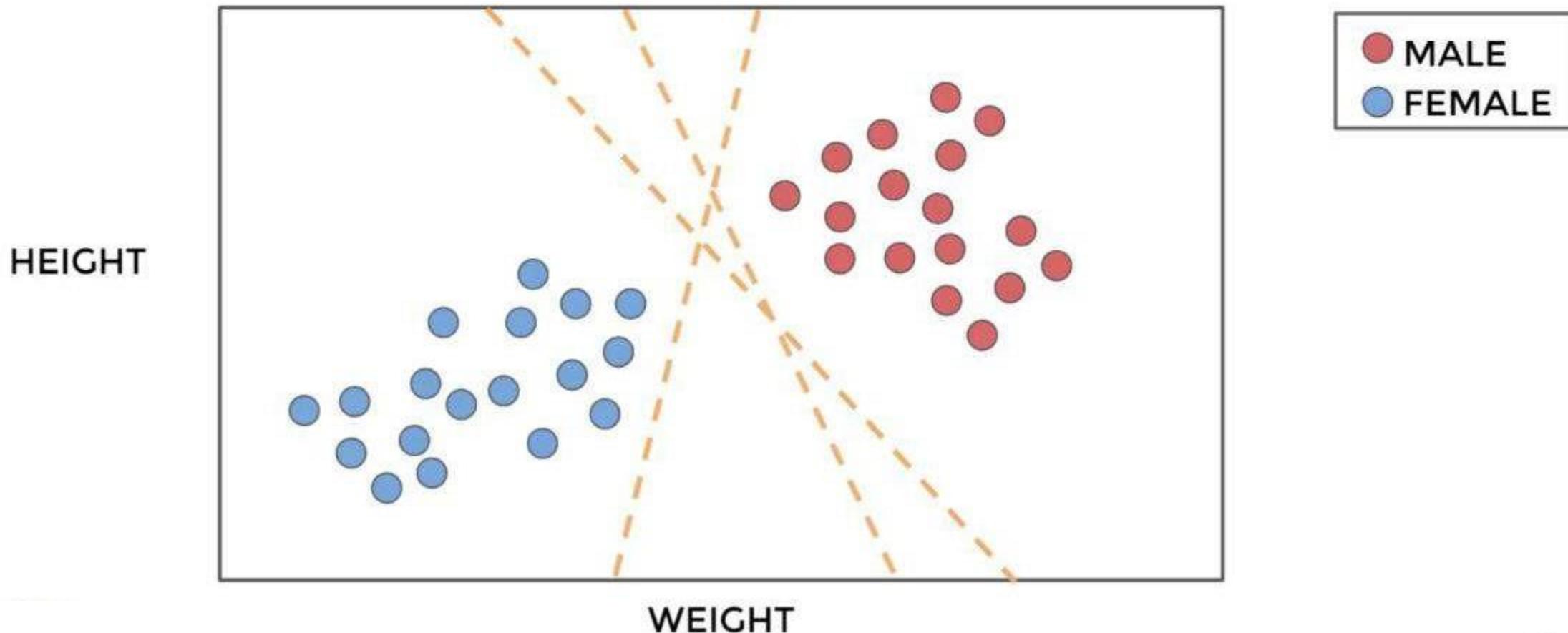
# Support Vector Machines

- Separating hyperplane is a line.



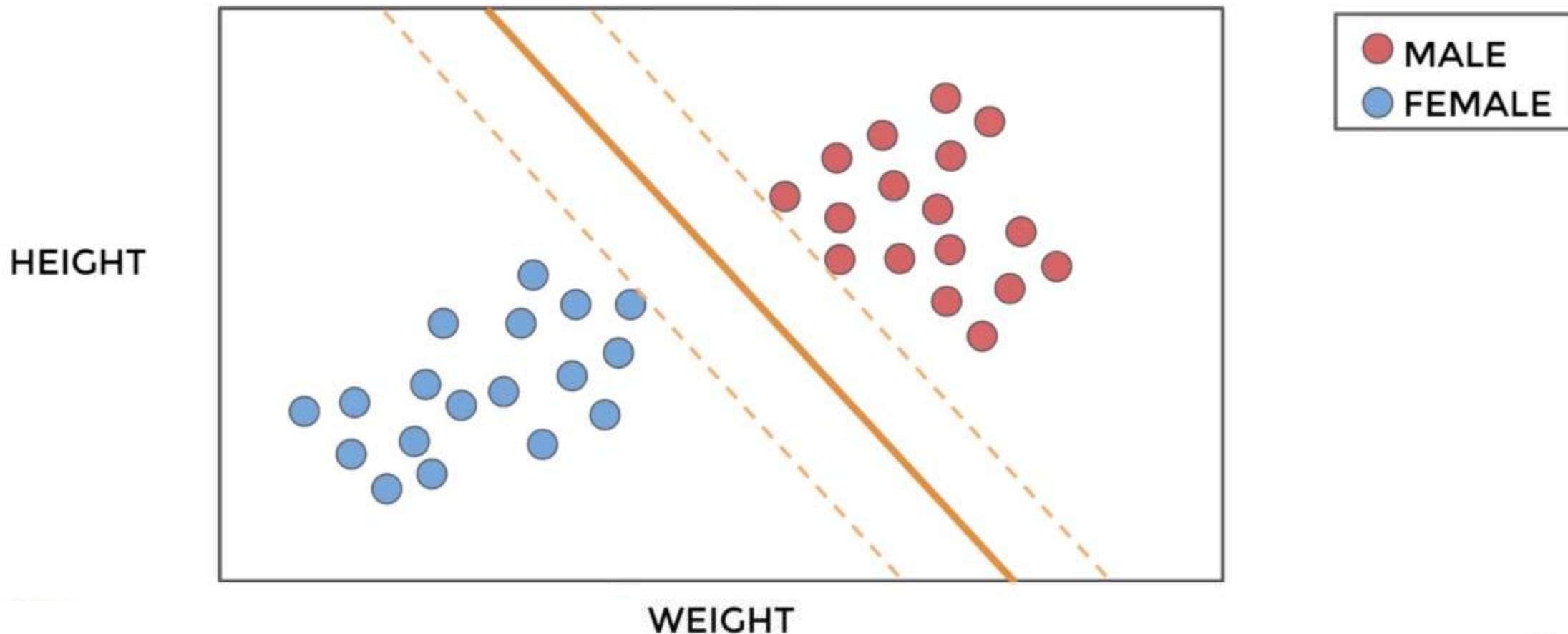
# Support Vector Machines

- Multiple possible hyperplanes:



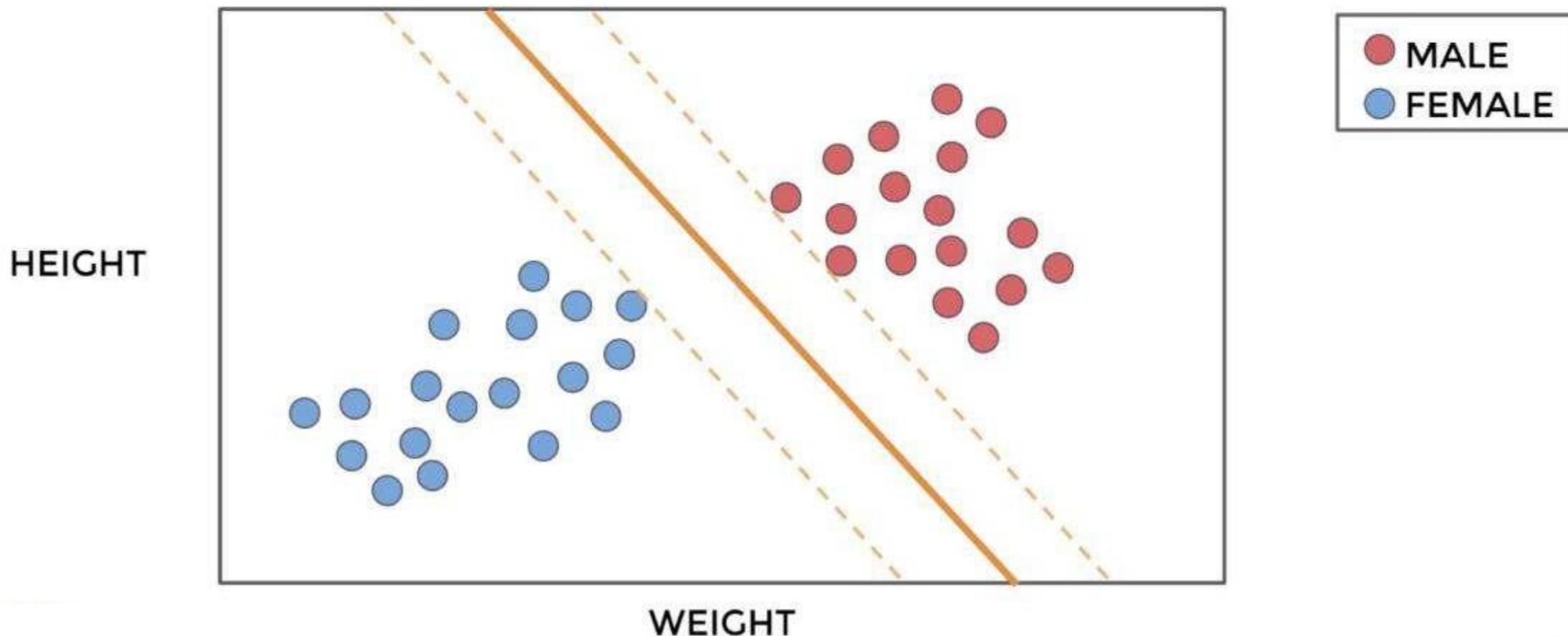
# Support Vector Machines

- Choose to maximize margins:



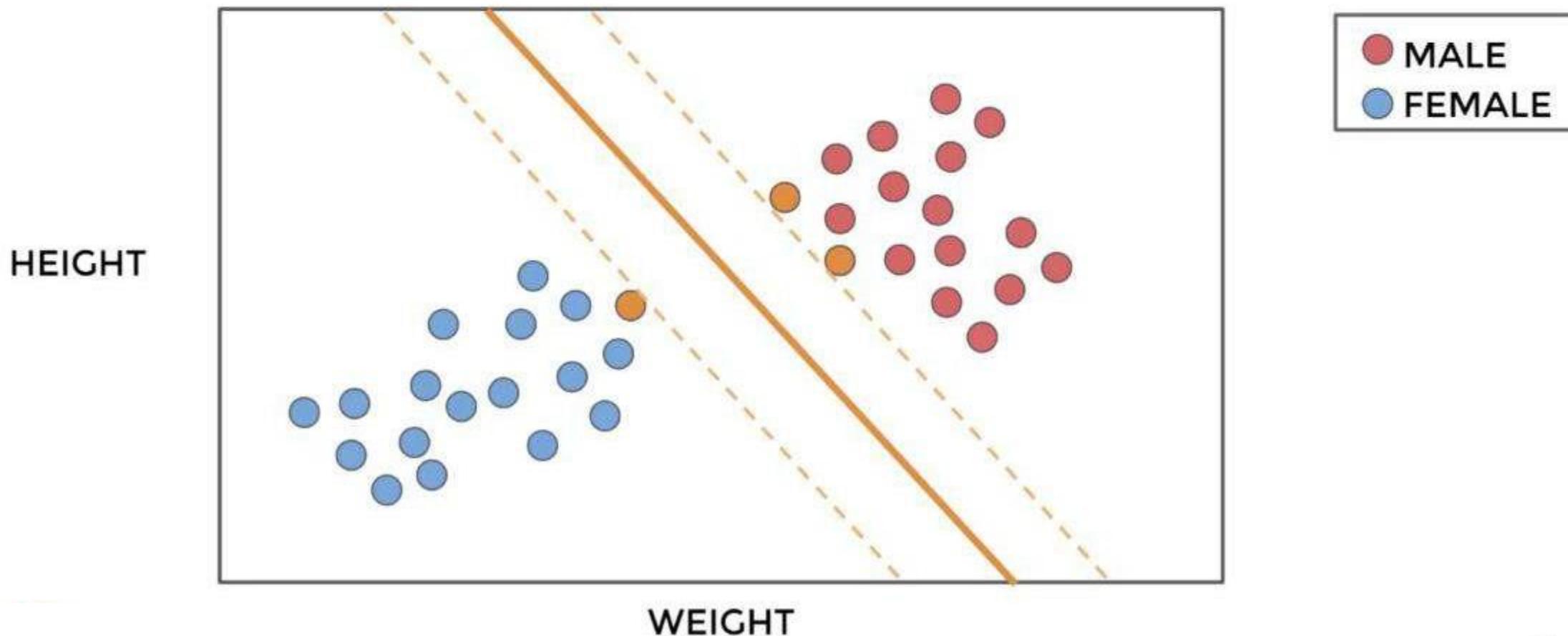
# Support Vector Machines

- Note each data point is a 2D vector:



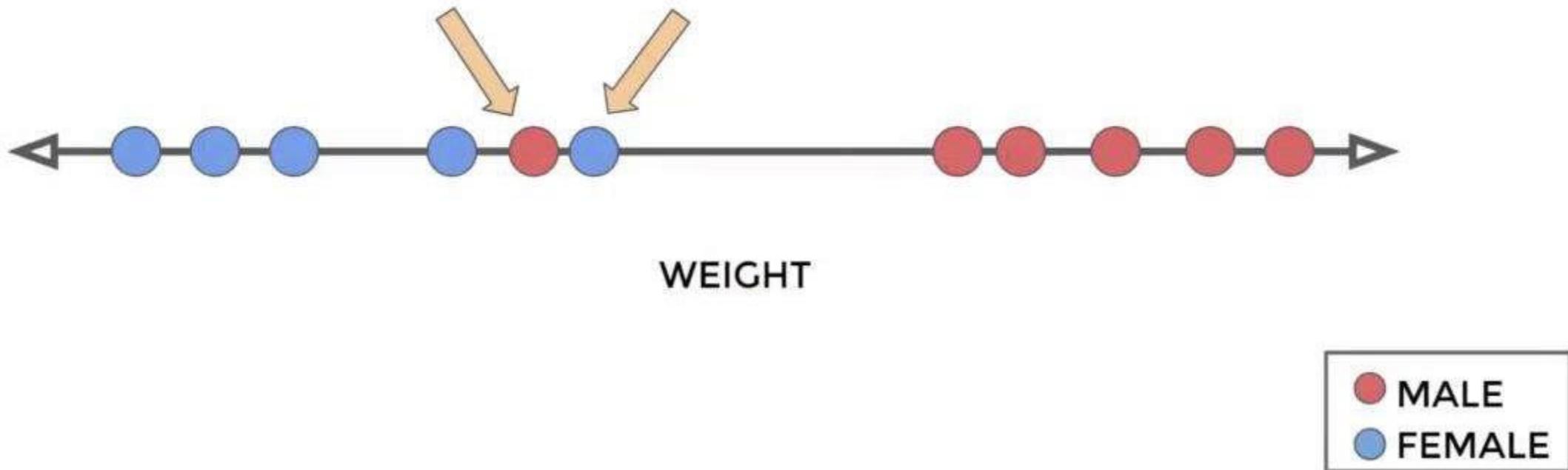
# Support Vector Machines

- Data points at margin “support” separator:



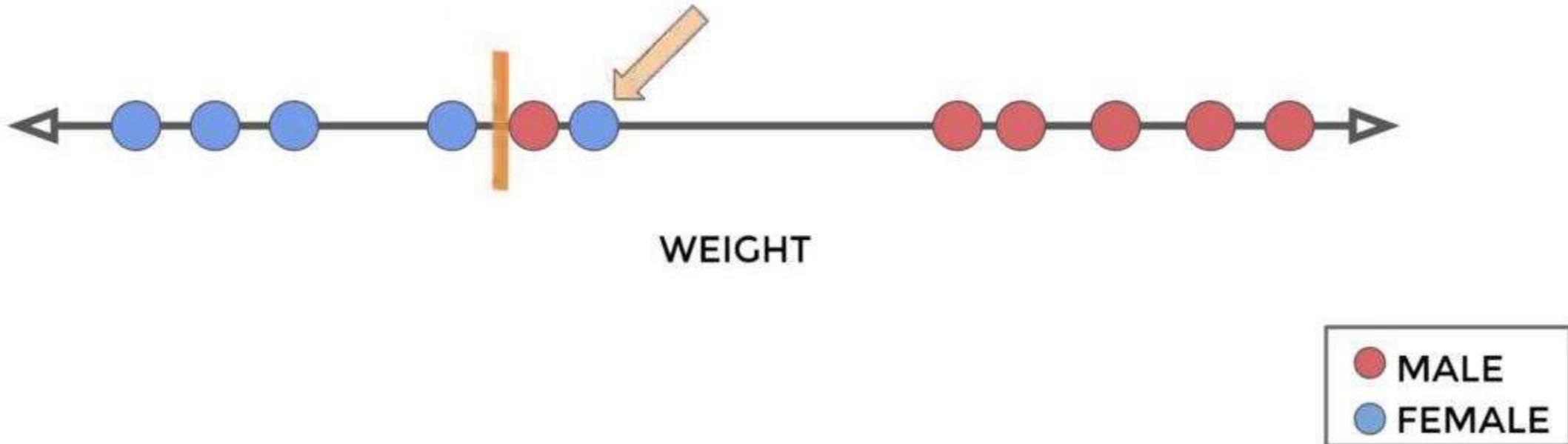
# Support Vector Machines

- What happens if classes are not perfectly separable?



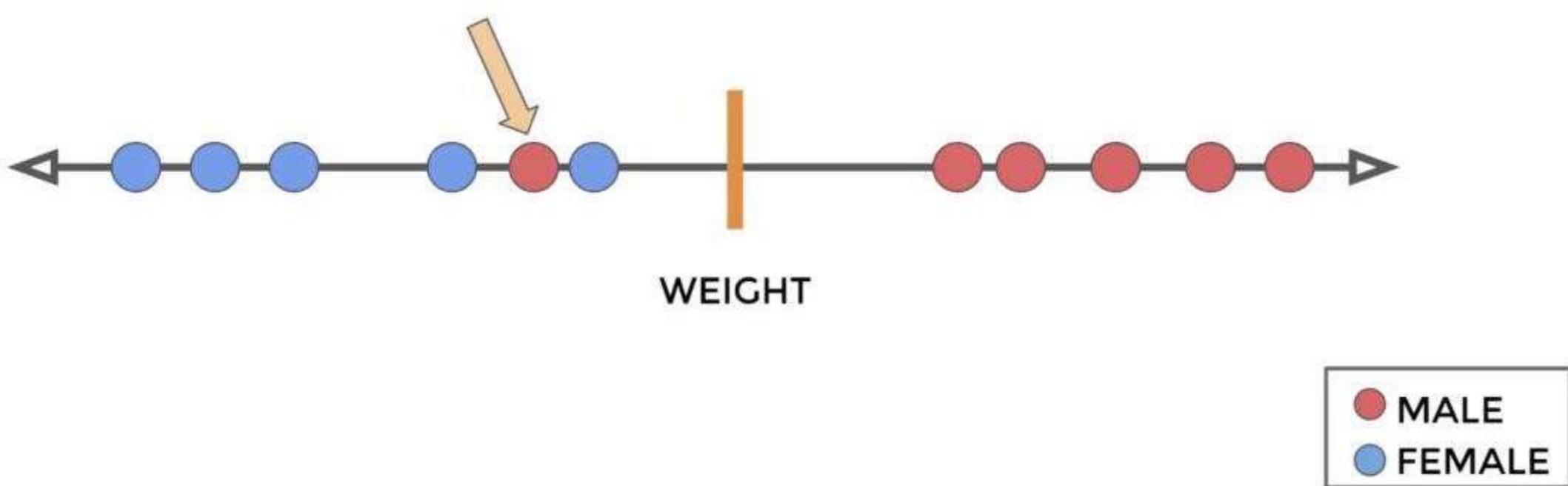
# Support Vector Machines

- We are not be able to separate without allowing for misclassifications.



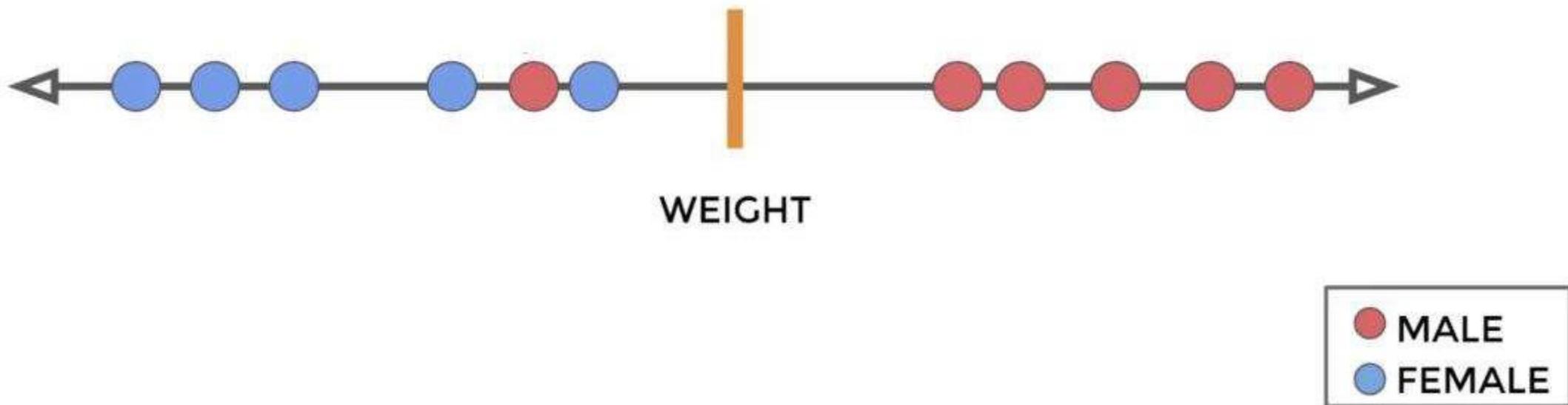
# Support Vector Machines

- We are not be able to separate without allowing for misclassifications.



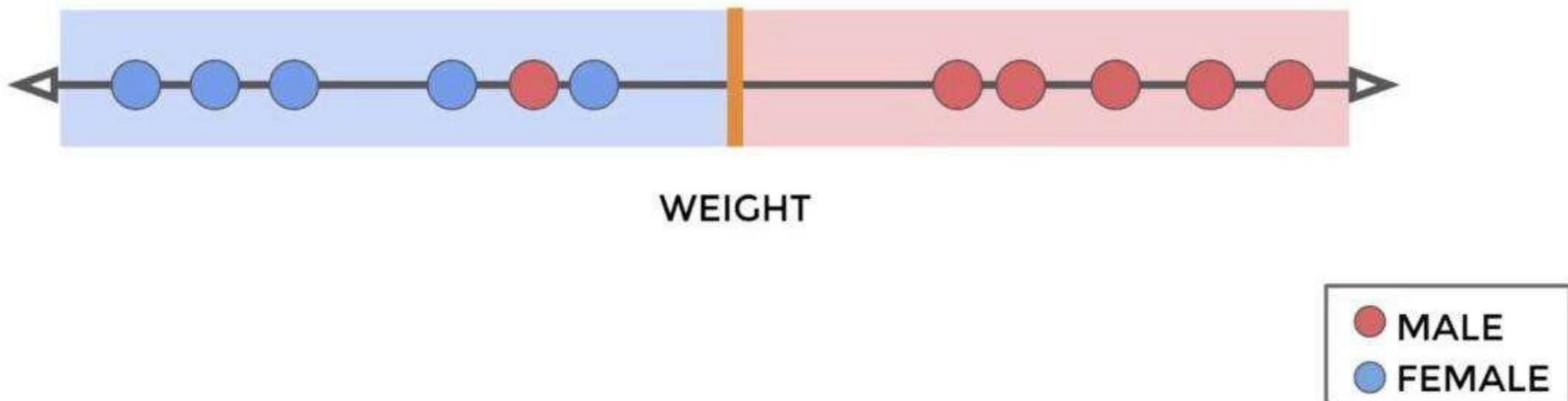
# Support Vector Machines

- We will have a bias-variance trade-off depending where we place this separator:



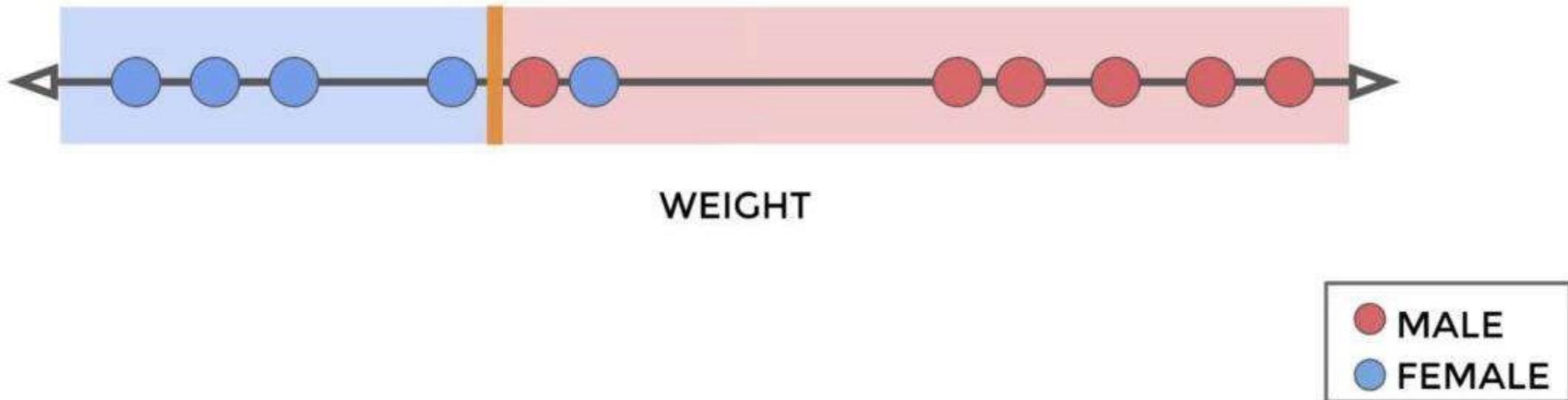
# Support Vector Machines

- For one feature this classifier creates ranges for male and female:



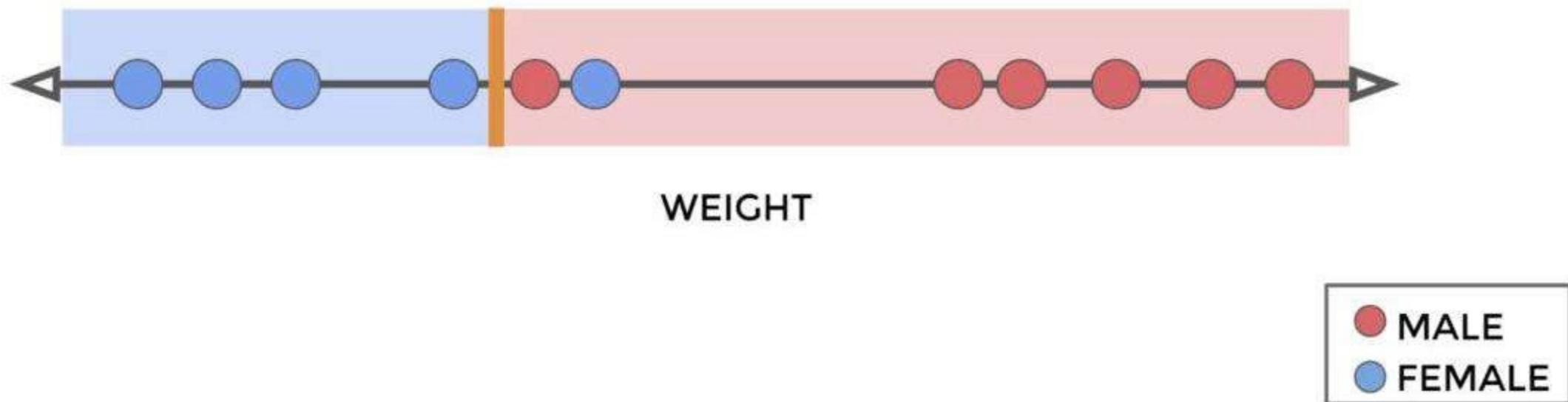
# Support Vector Machines

- This fit only misclassified one female training point as male:



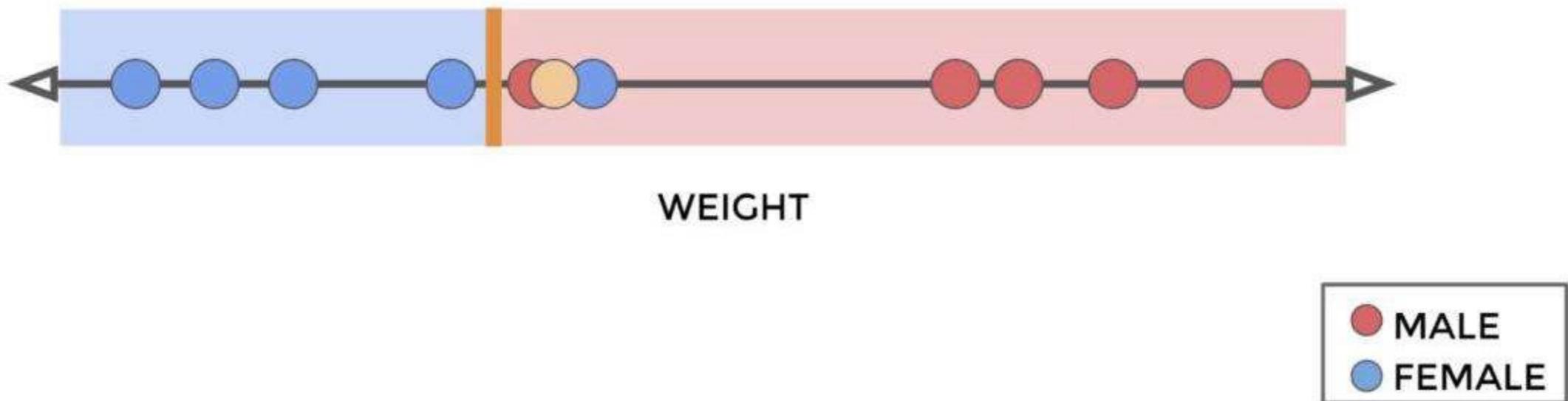
# Support Vector Machines

- This looks like a high variance fit to training data, picking too much noise from Female:



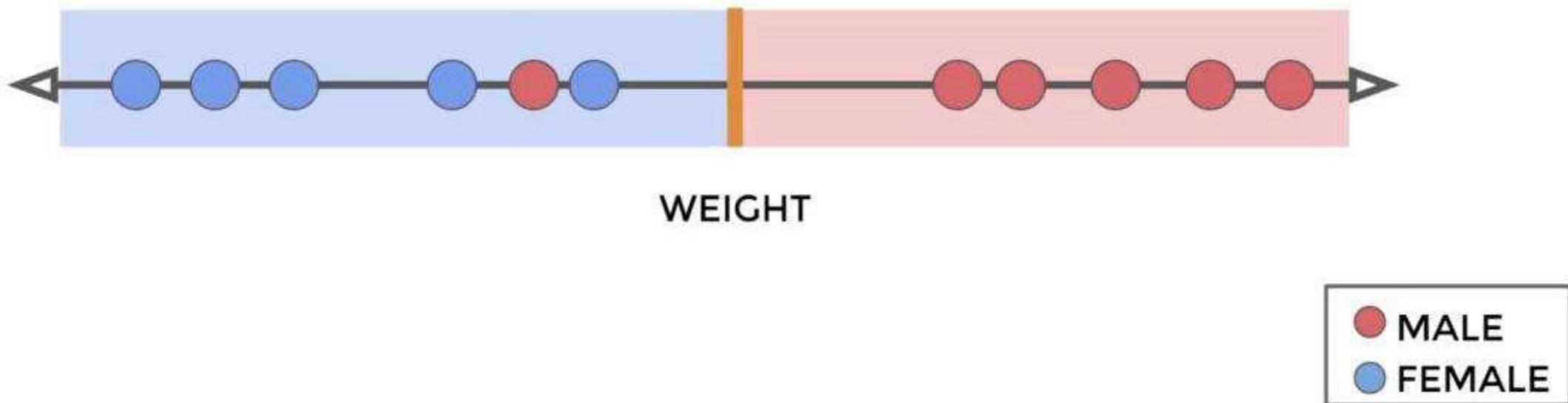
# Support Vector Machines

- A new test point close to existing female weights could get classified as male:



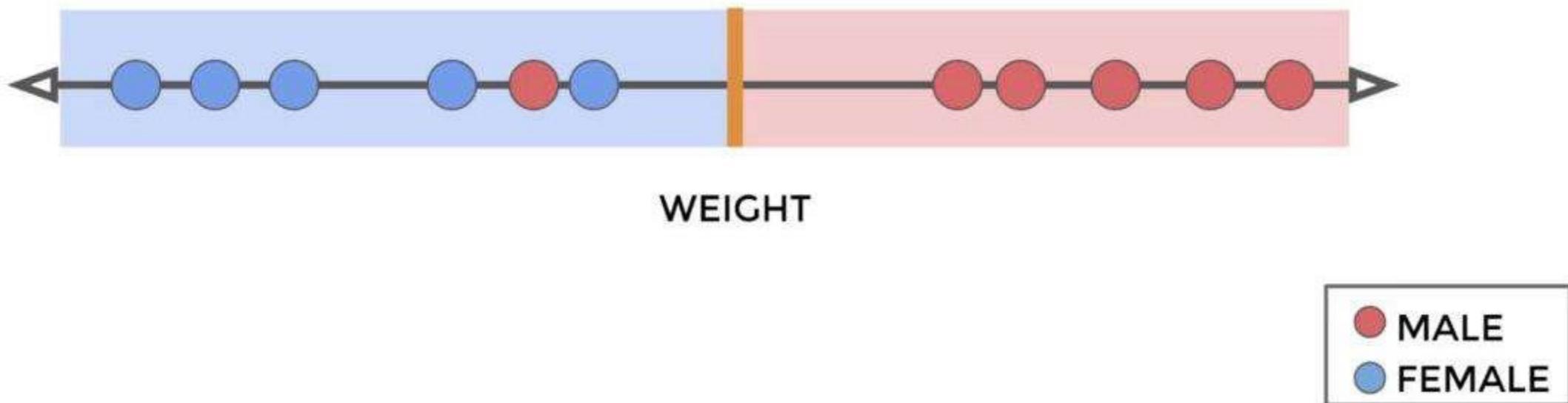
# Support Vector Machines

- We will have a bias-variance trade-off depending where we place this separator:



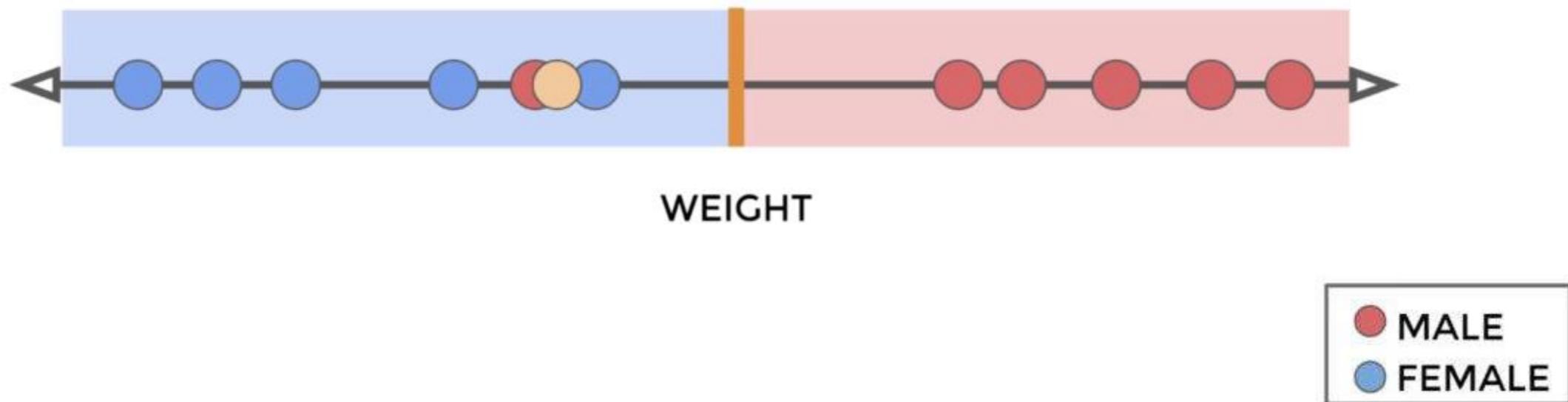
# Support Vector Machines

- Here we allow more bias to lead to better long term results on future data:



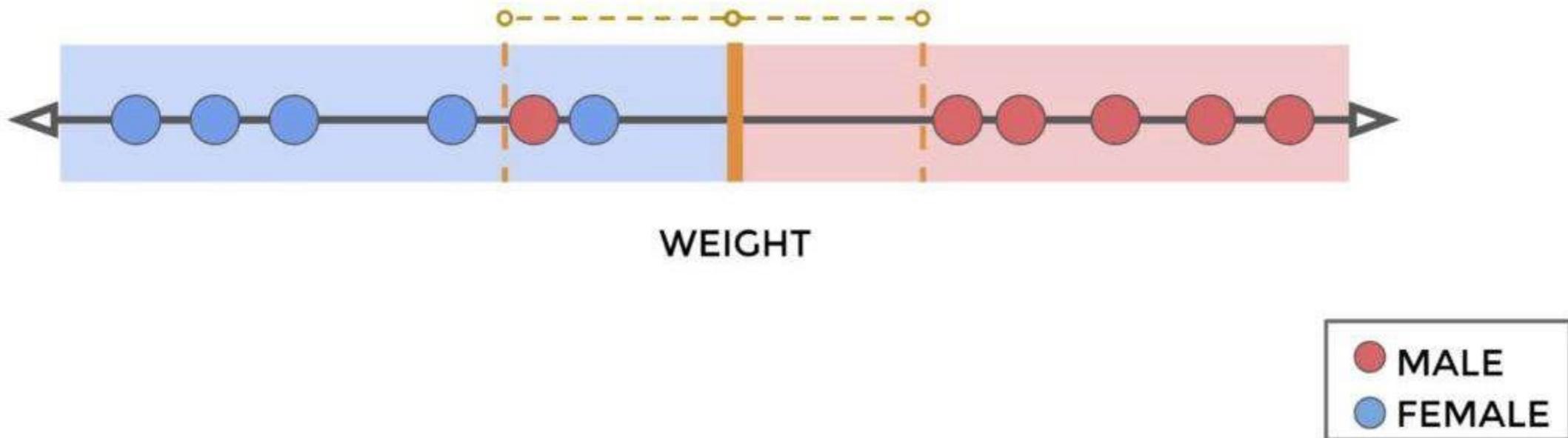
# Support Vector Machines

- Here we allow more bias to lead to better long term results on future data:



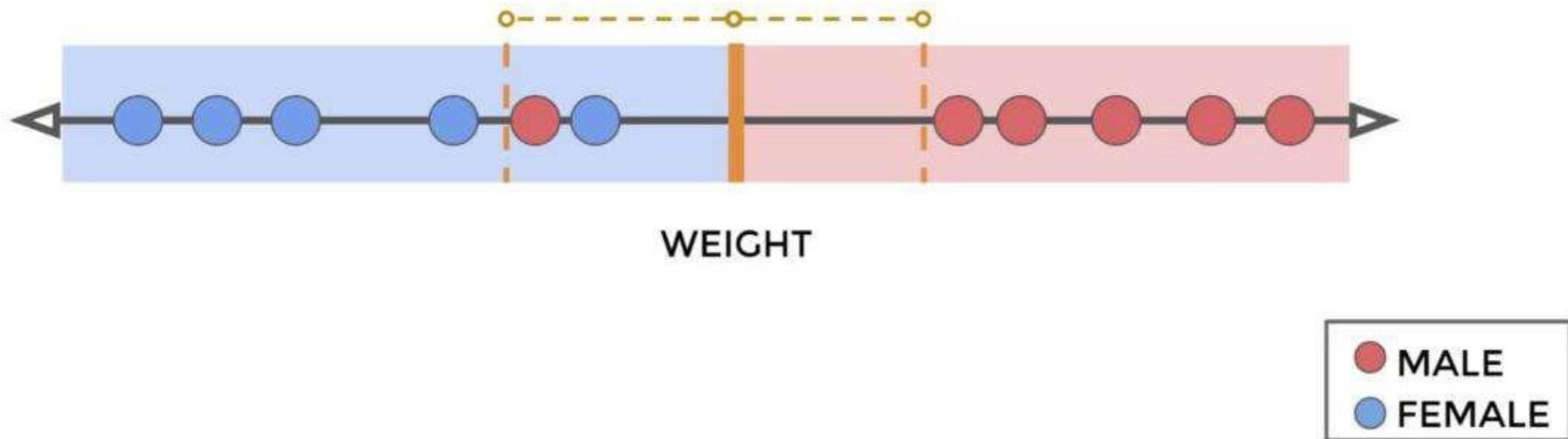
# Support Vector Machines

- Distance between threshold and the observations is a **soft margin**:



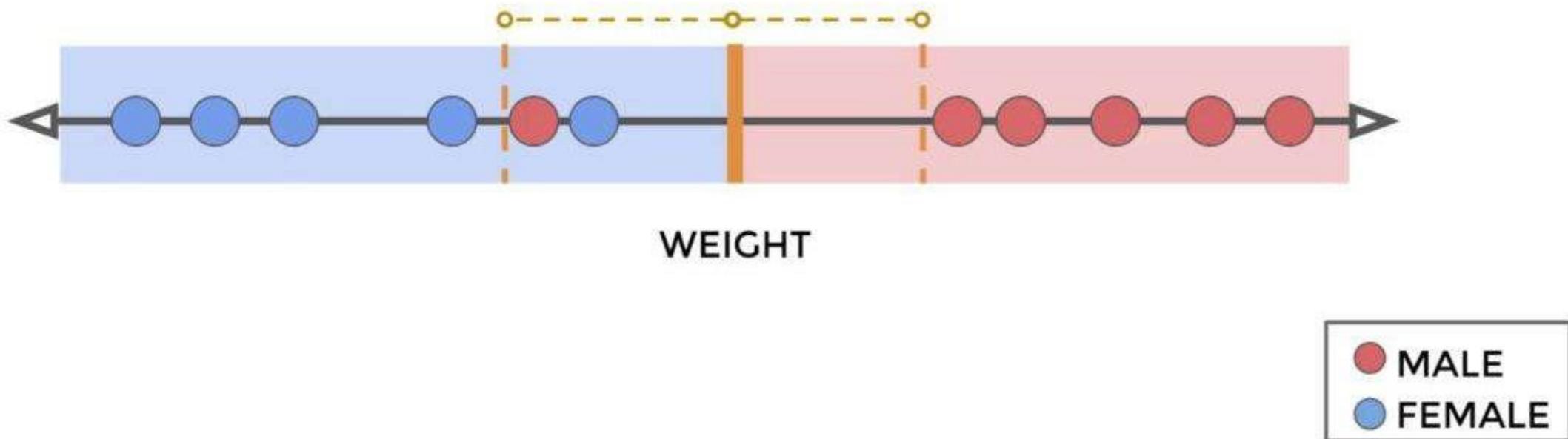
# Support Vector Machines

- Soft margin allows for misclassification inside the margins.



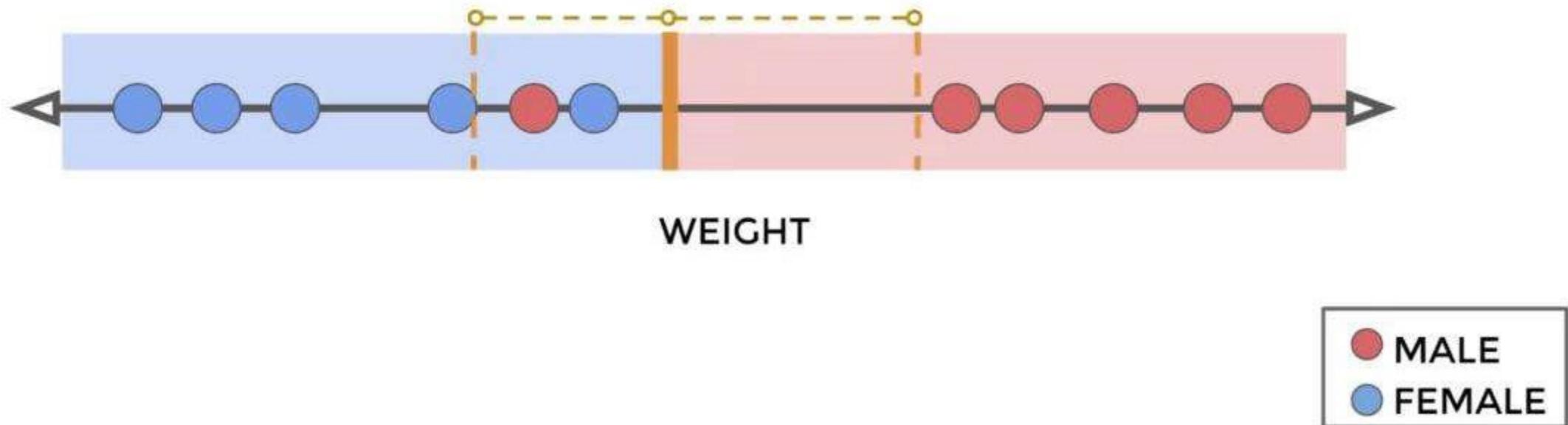
# Support Vector Machines

- There are many possible threshold splits if we allow for soft margins.



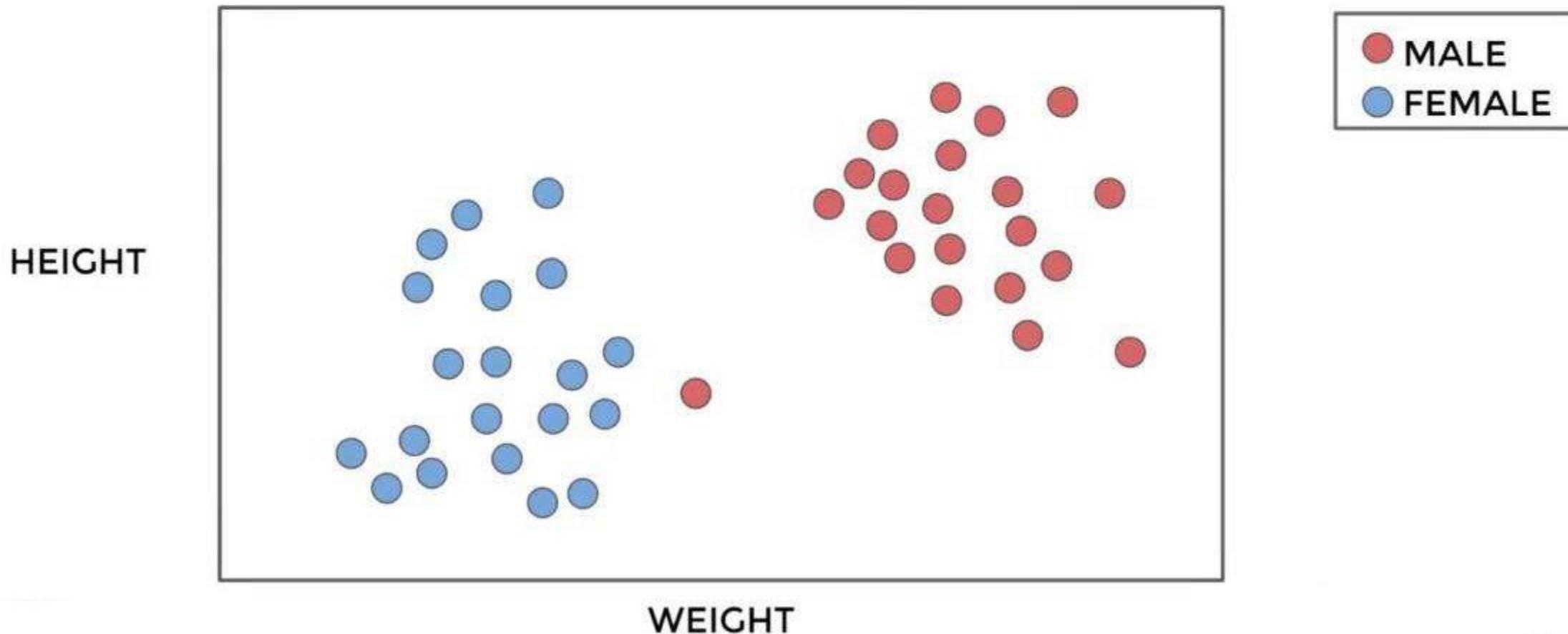
# Support Vector Machines

- We can use cross validation to determine the optimal size of the margins.



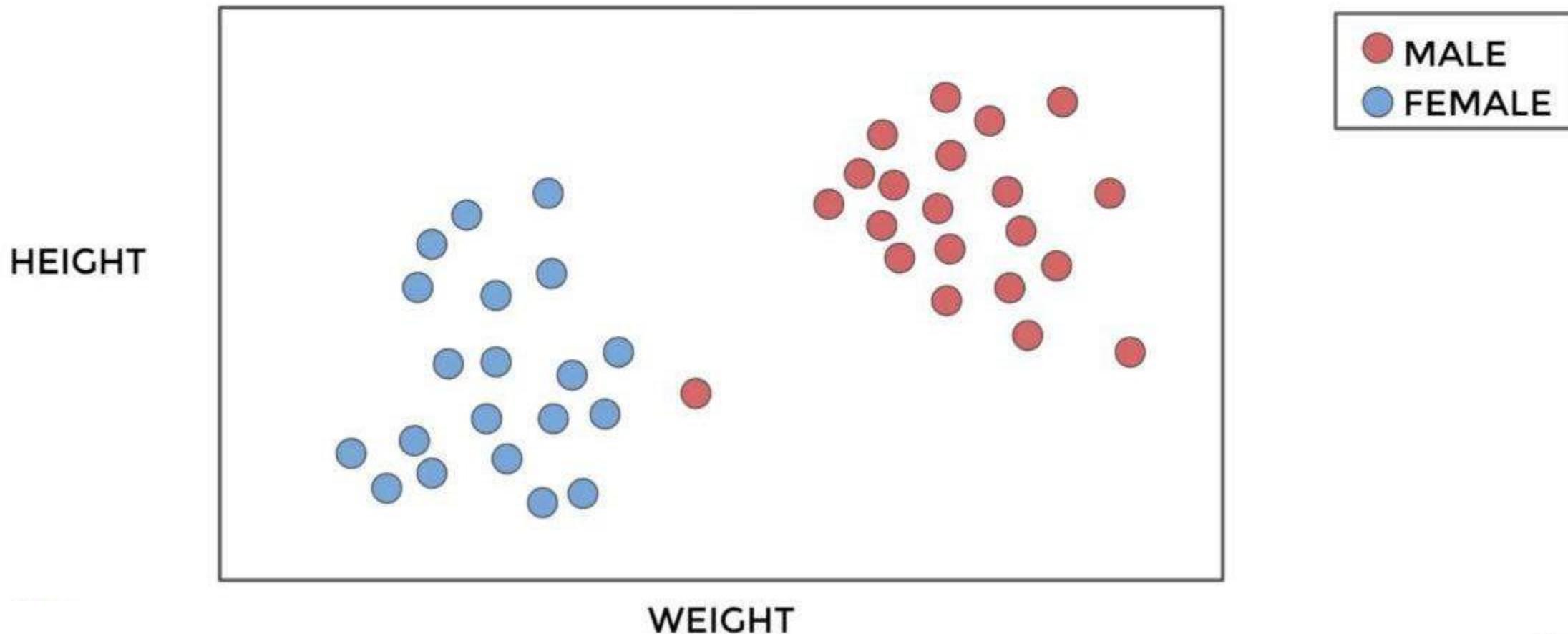
# Support Vector Machines

- 2D soft margin example:



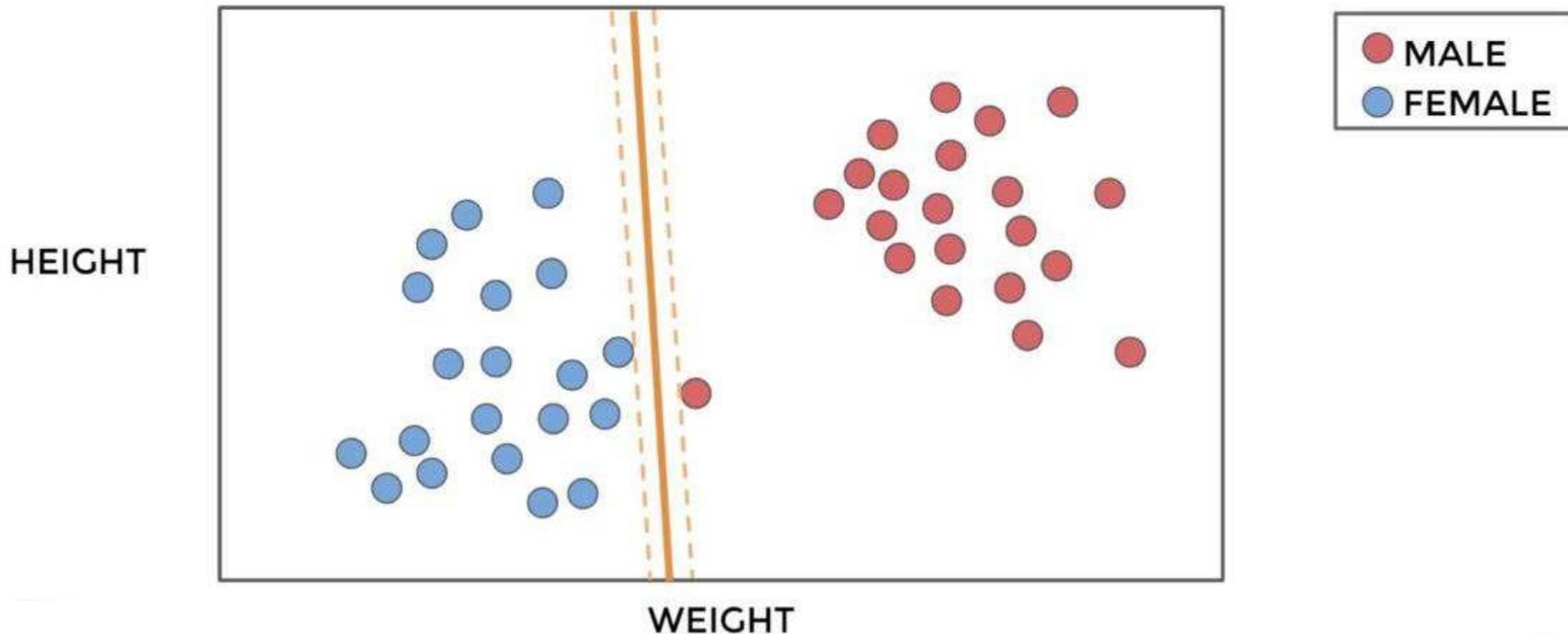
# Support Vector Machines

- Data set is technically perfectly separable



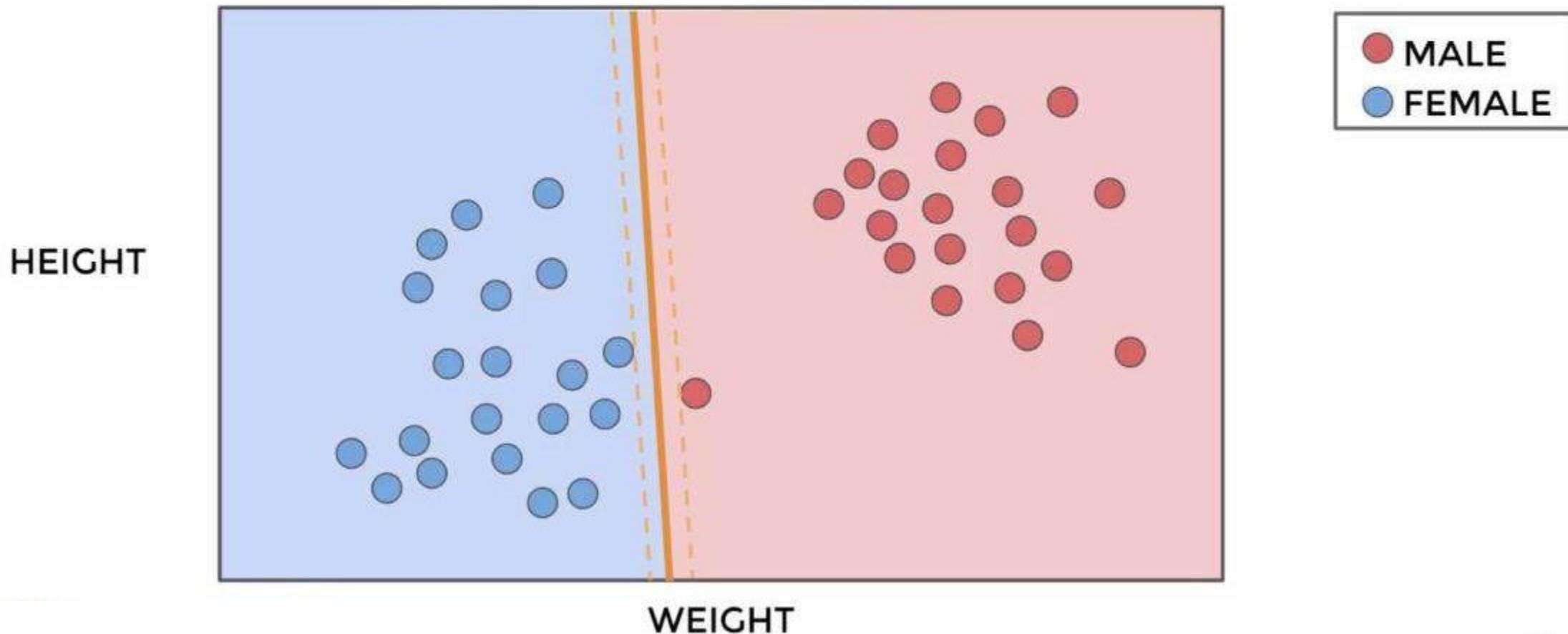
# Support Vector Machines

- Maximal Margin Classifier



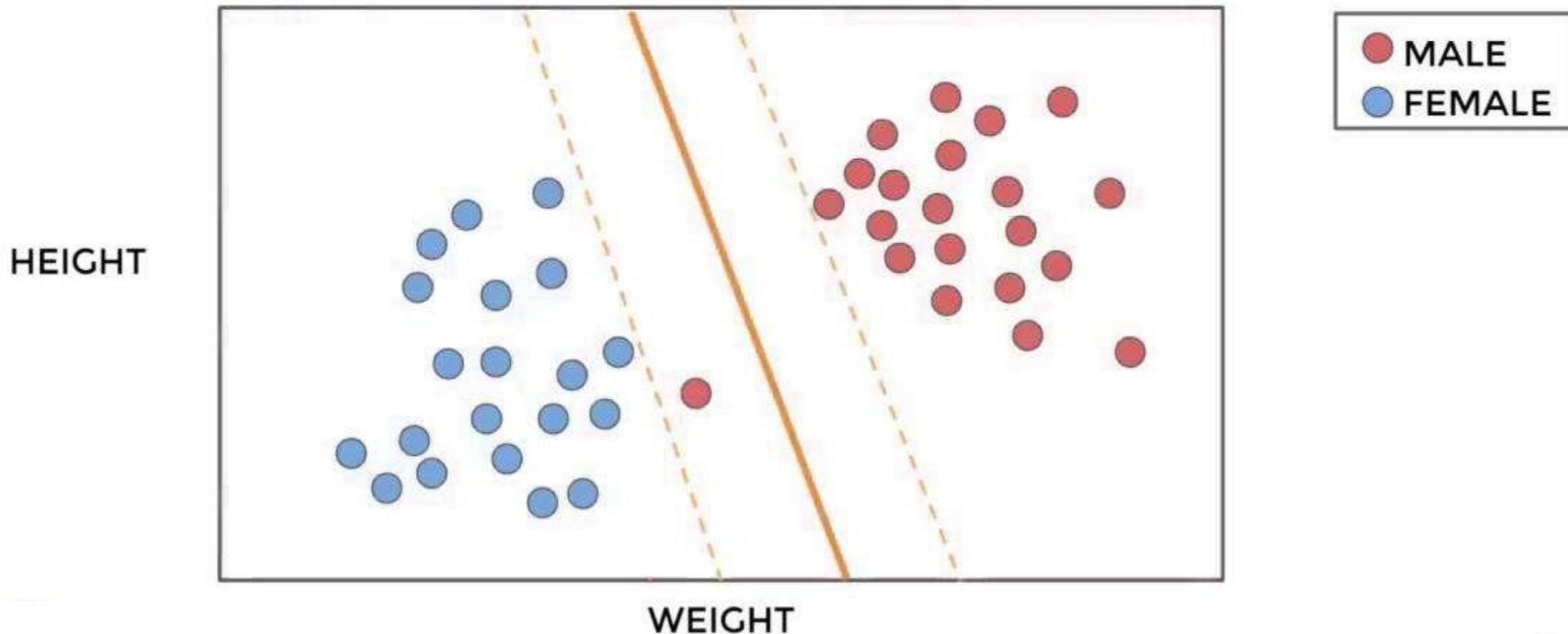
# Support Vector Machines

- Maximal Margin Classifier



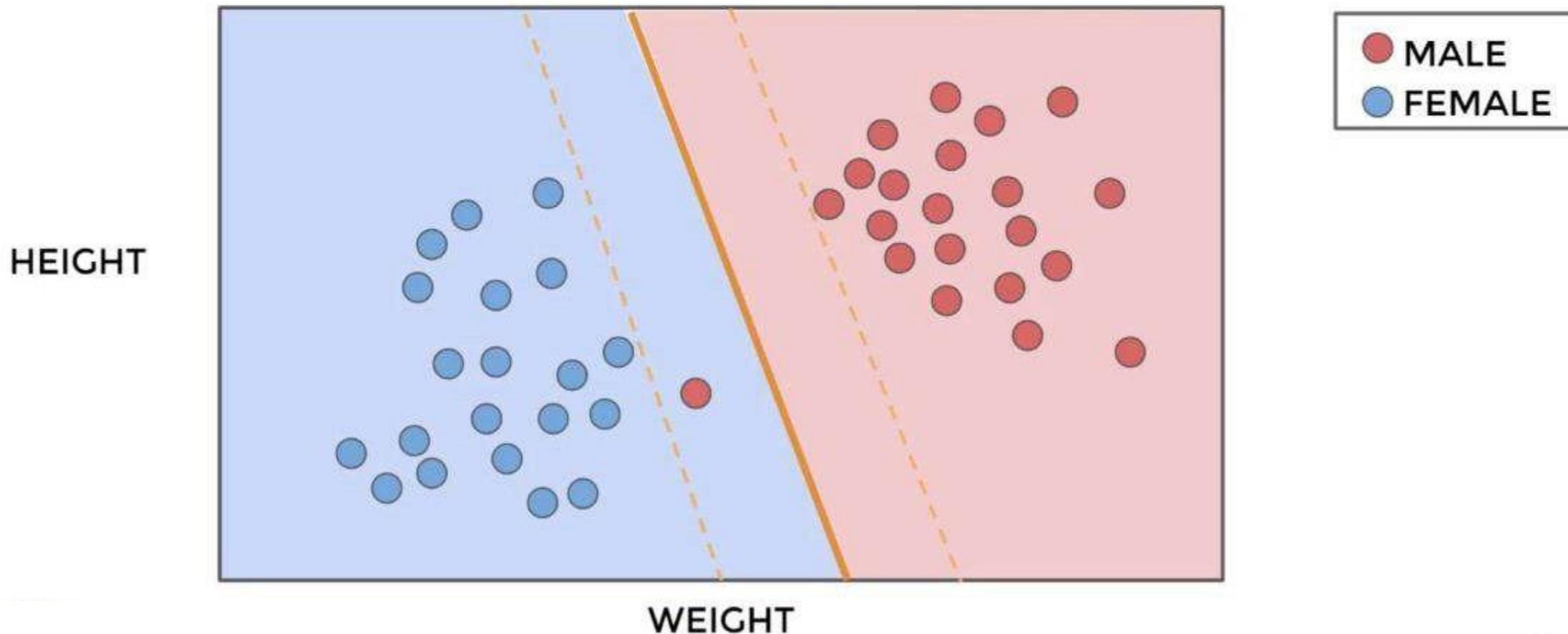
# Support Vector Machines

- Support Vector Classifier (Soft Margins)



# Support Vector Machines

- Support Vector Classifier (Soft Margins)



# Support Vector Machines

- We've only visualized cases where the classes are easily separated by the hyperplane in the original feature space.
- Allowing for some misclassifications still resulted in reasonable results.
- What would happen in a case where a hyperplane performs poorly, even when allowing for misclassifications?

# Support Vector Machines

- Notice a single hyperplane won't separate out the classes without many misclassifications!

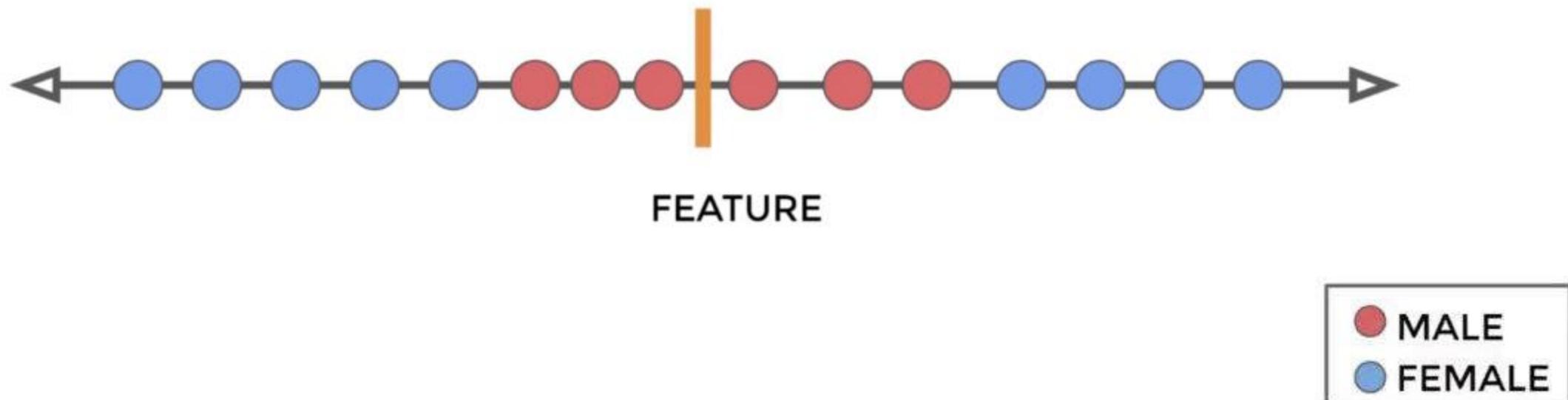


FEATURE



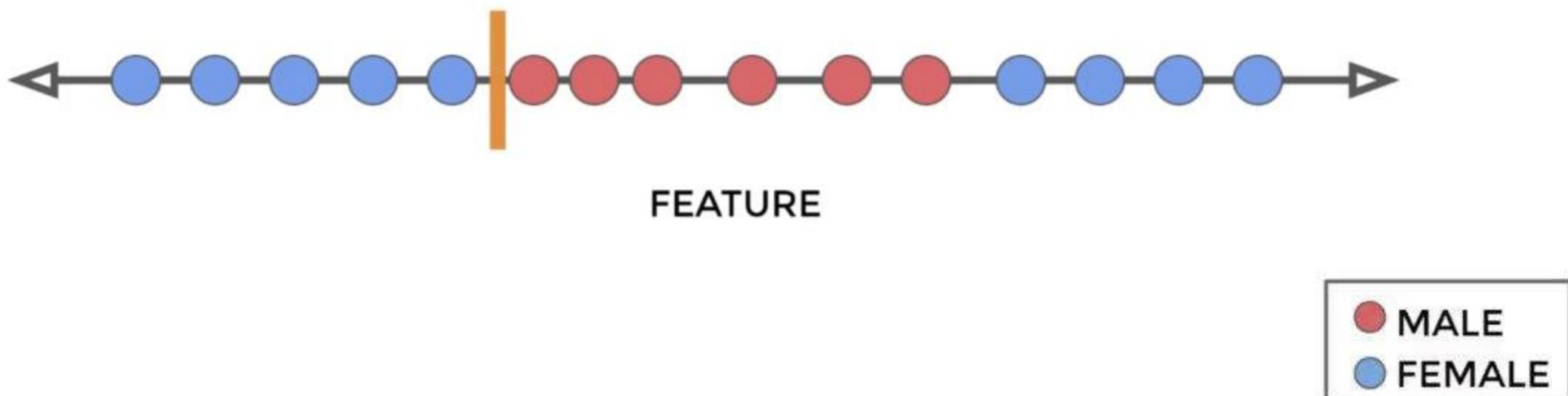
# Support Vector Machines

- Notice a single hyperplane won't separate out the classes without many misclassifications!



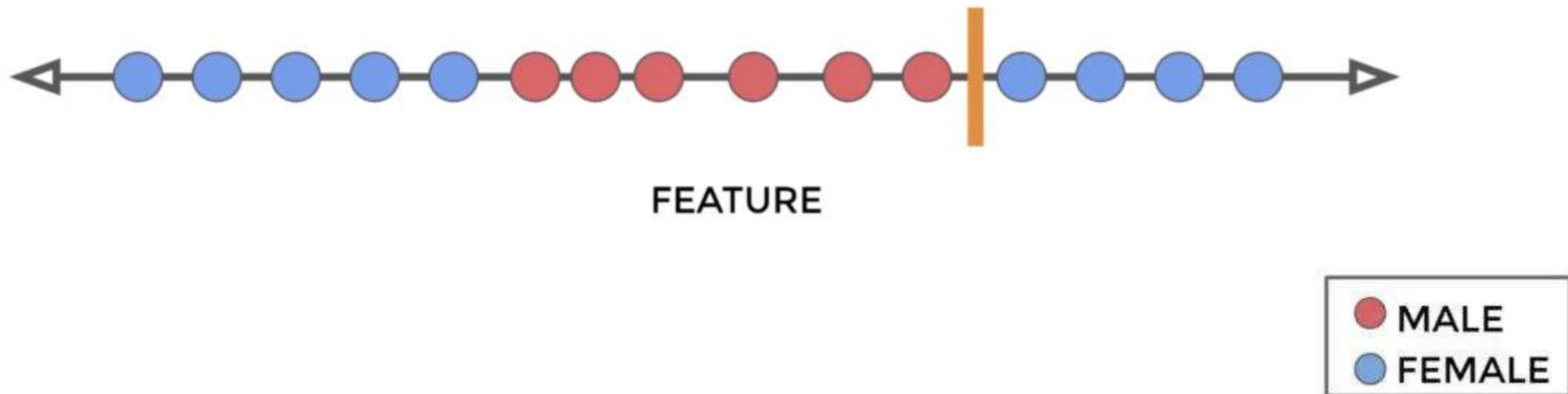
# Support Vector Machines

- Notice a single hyperplane won't separate out the classes without many misclassifications!



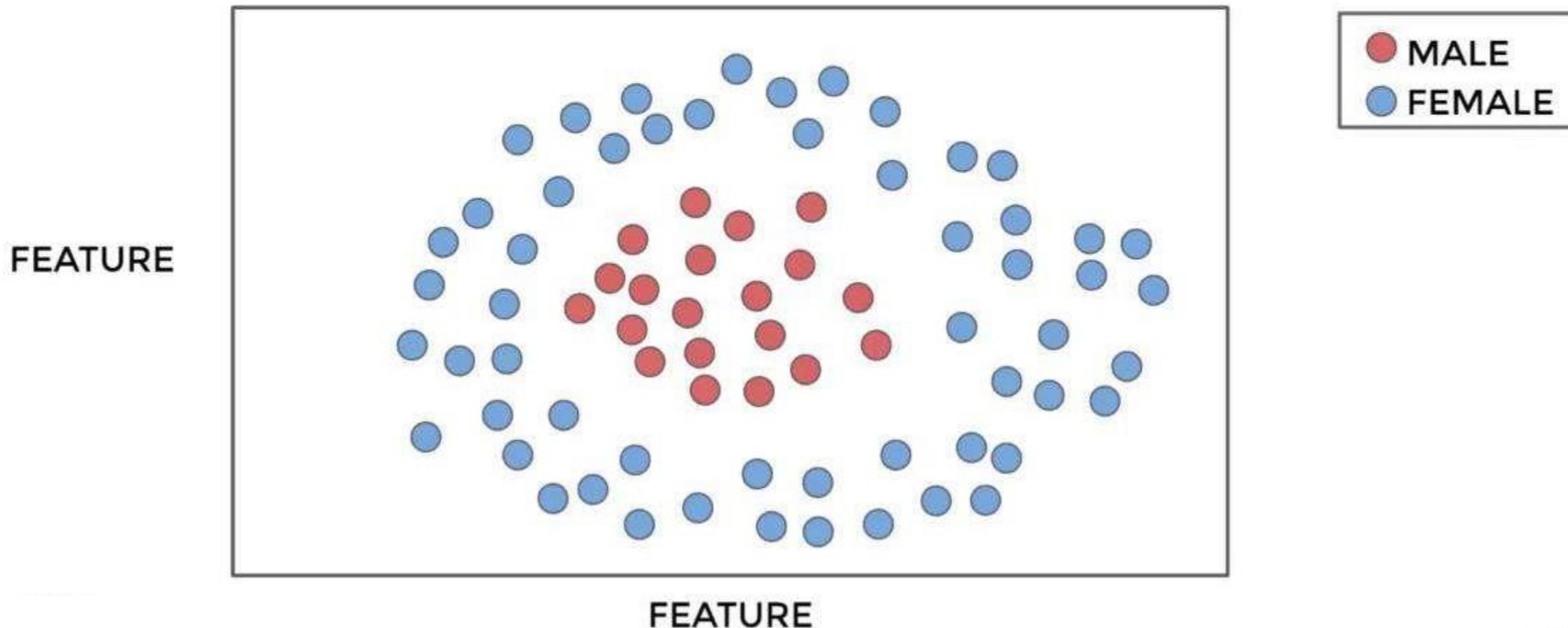
# Support Vector Machines

- Notice a single hyperplane won't separate out the classes without many misclassifications!



# Support Vector Machines

- Can't split classes with hyperplane line:



# Support Vector Machines

- To solve these cases, we move on from Support Vector Classifier, to Support Vector Machines.
- SVMs use **kernels** to project the data to a higher dimension, in order to use a hyperplane in this higher dimension to separate the data.

# Support Vector Machines

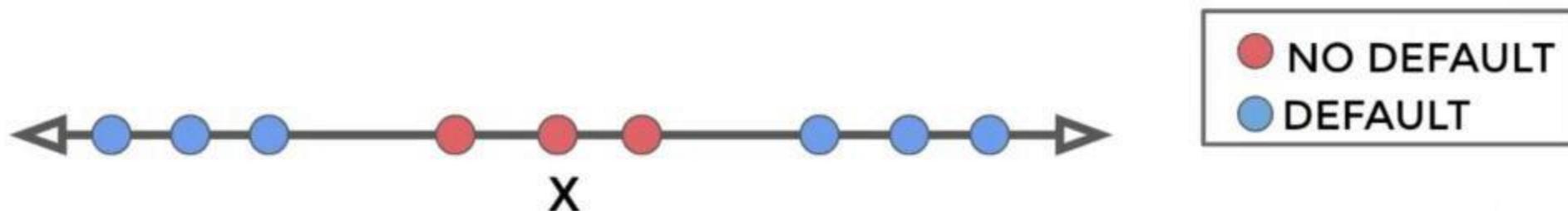
Theory and Intuition - Kernels

# Support Vector Machines

- Kernels allow us to move beyond a Support Vector Classifier and use Support Vector Machines.
- There are a variety of kernels we can use to “project” the features to a higher dimension.
- Let’s explore how this works through some visual examples...

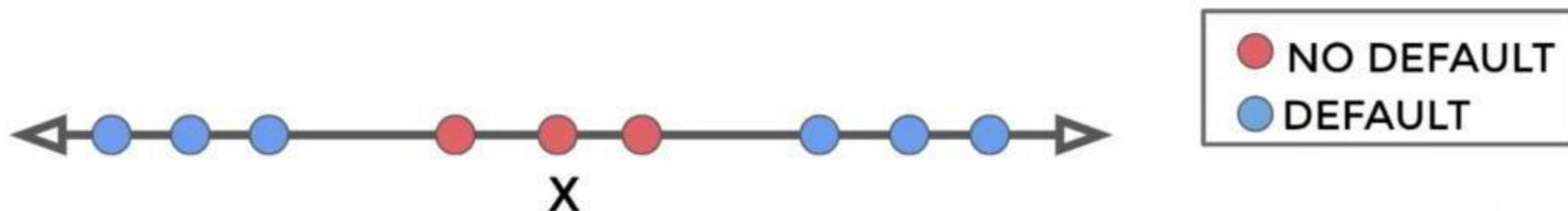
# Support Vector Machines

- Recall our 1D example of classes not easily separated by a single hyperplane:



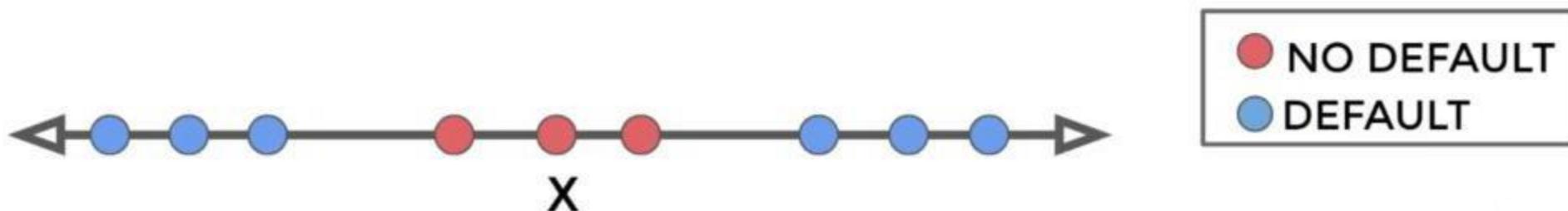
# Support Vector Machines

- Let's explore how using a kernel could project this feature onto another dimension.



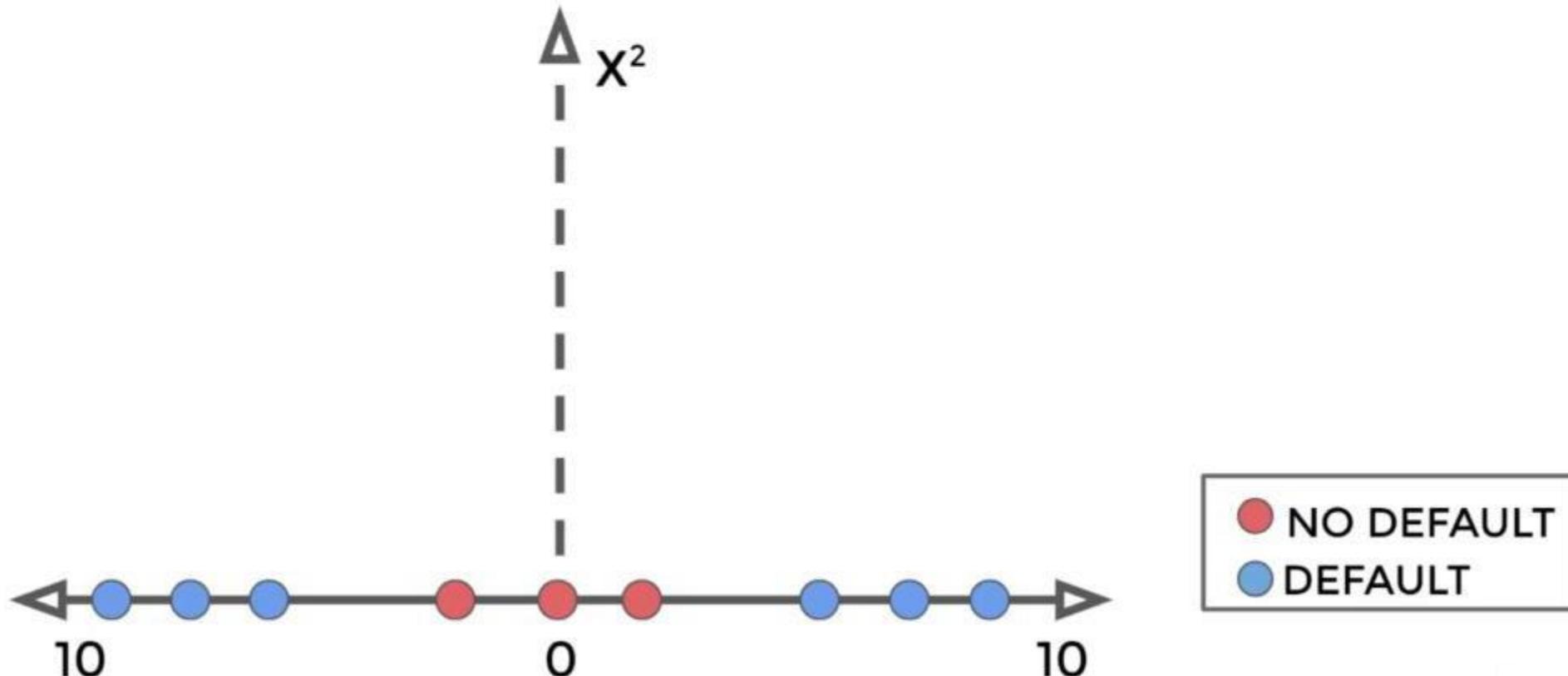
# Support Vector Machines

- For example, a polynomial kernel could expand onto an  $X^2$  dimension:



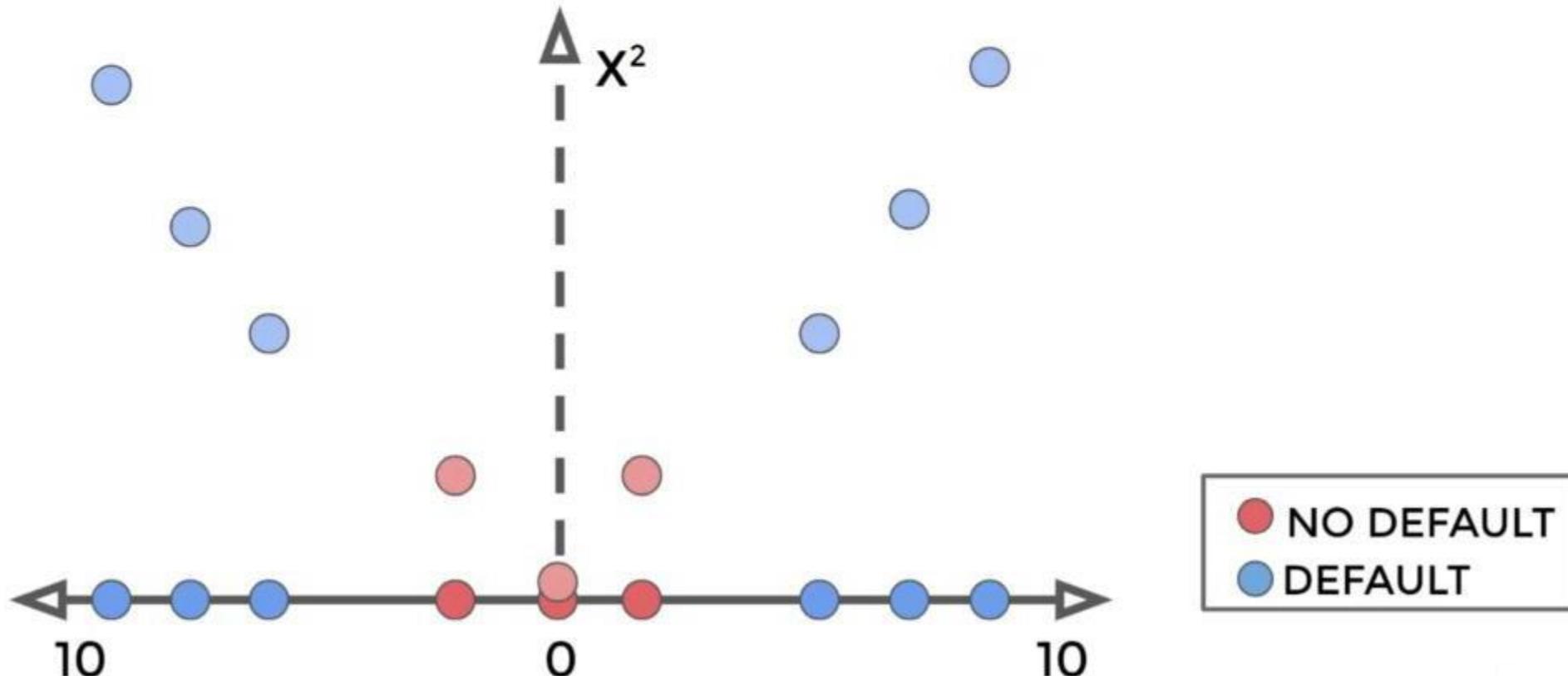
# Support Vector Machines

- For example, a polynomial kernel could expand onto an  $X^2$  dimension:



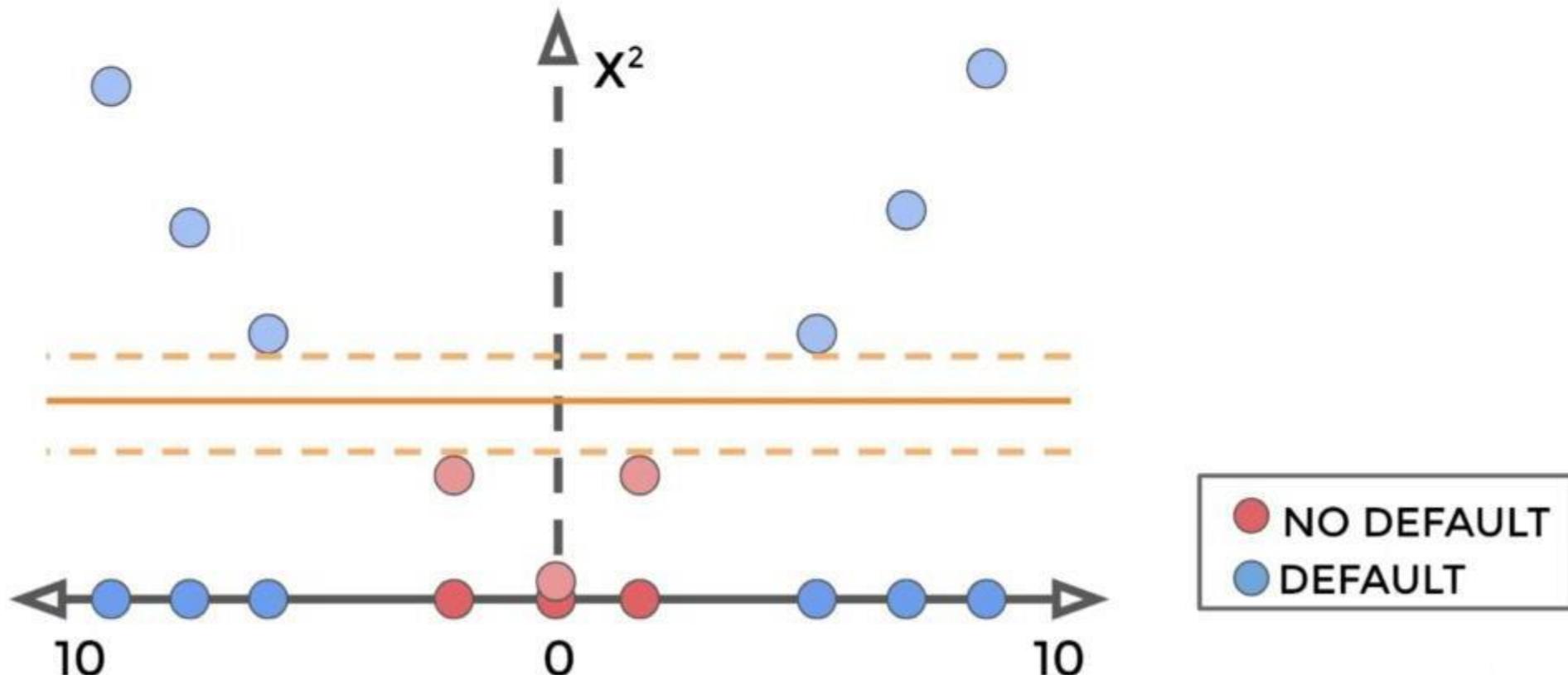
# Support Vector Machines

- For example, a polynomial kernel could expand onto an  $X^2$  dimension:



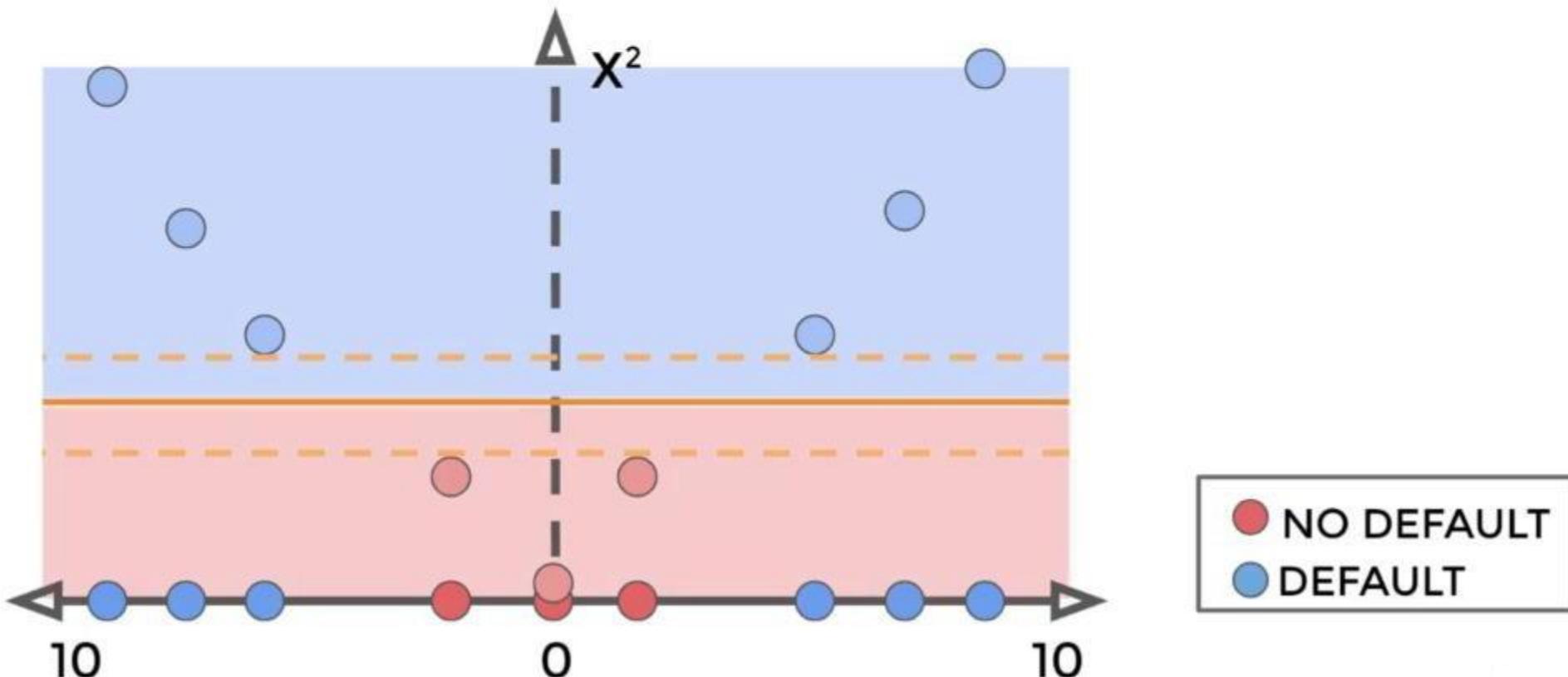
# Support Vector Machines

- Create a hyperplane after this projection:



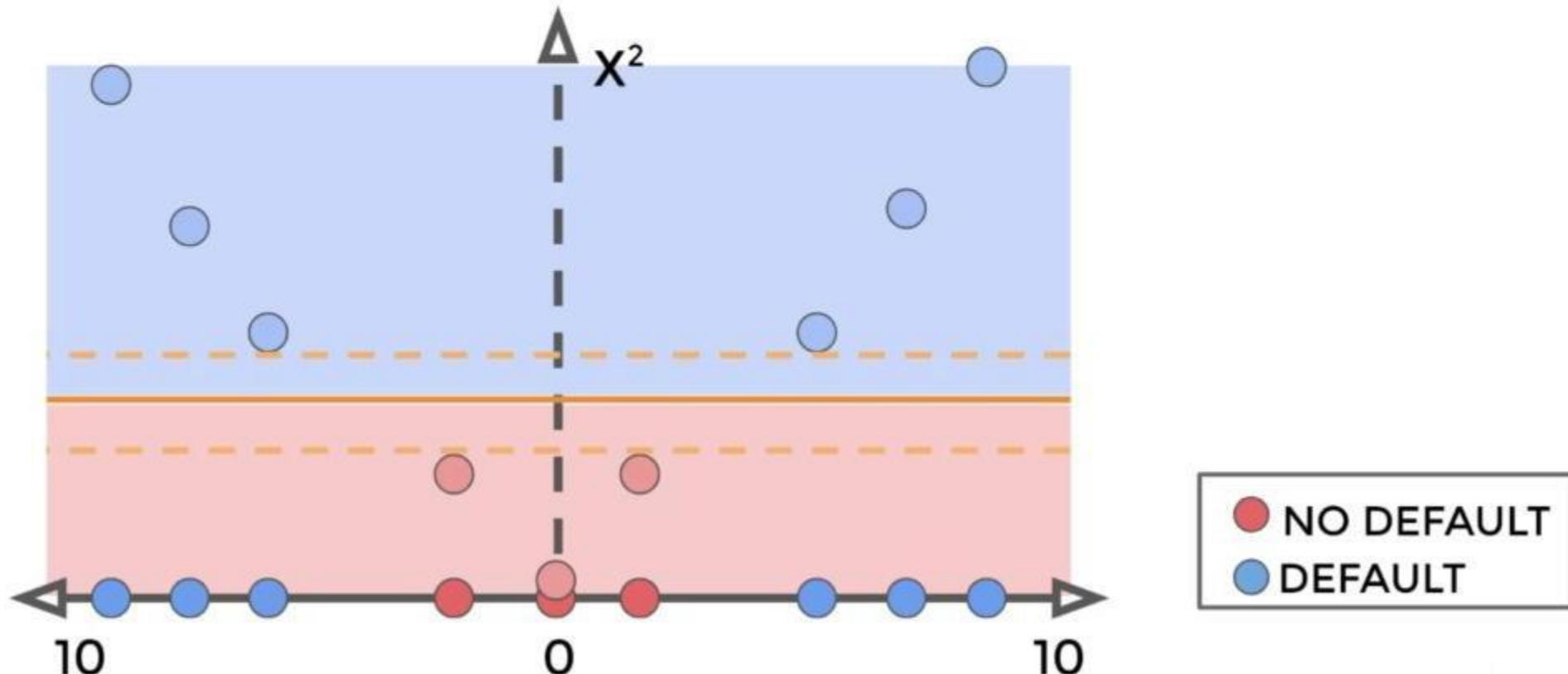
# Support Vector Machines

- Create a hyperplane after this projection:



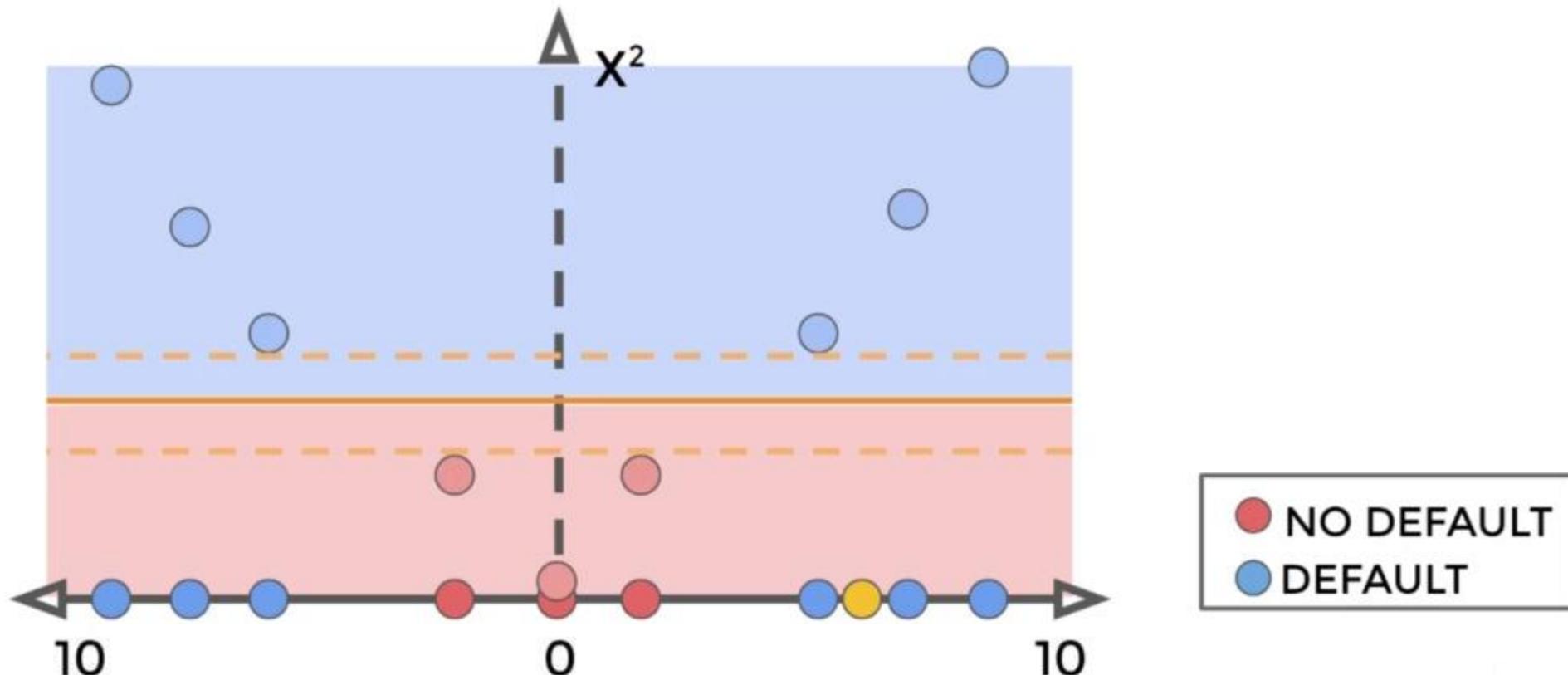
# Support Vector Machines

- Use kernel projection to evaluate new points:



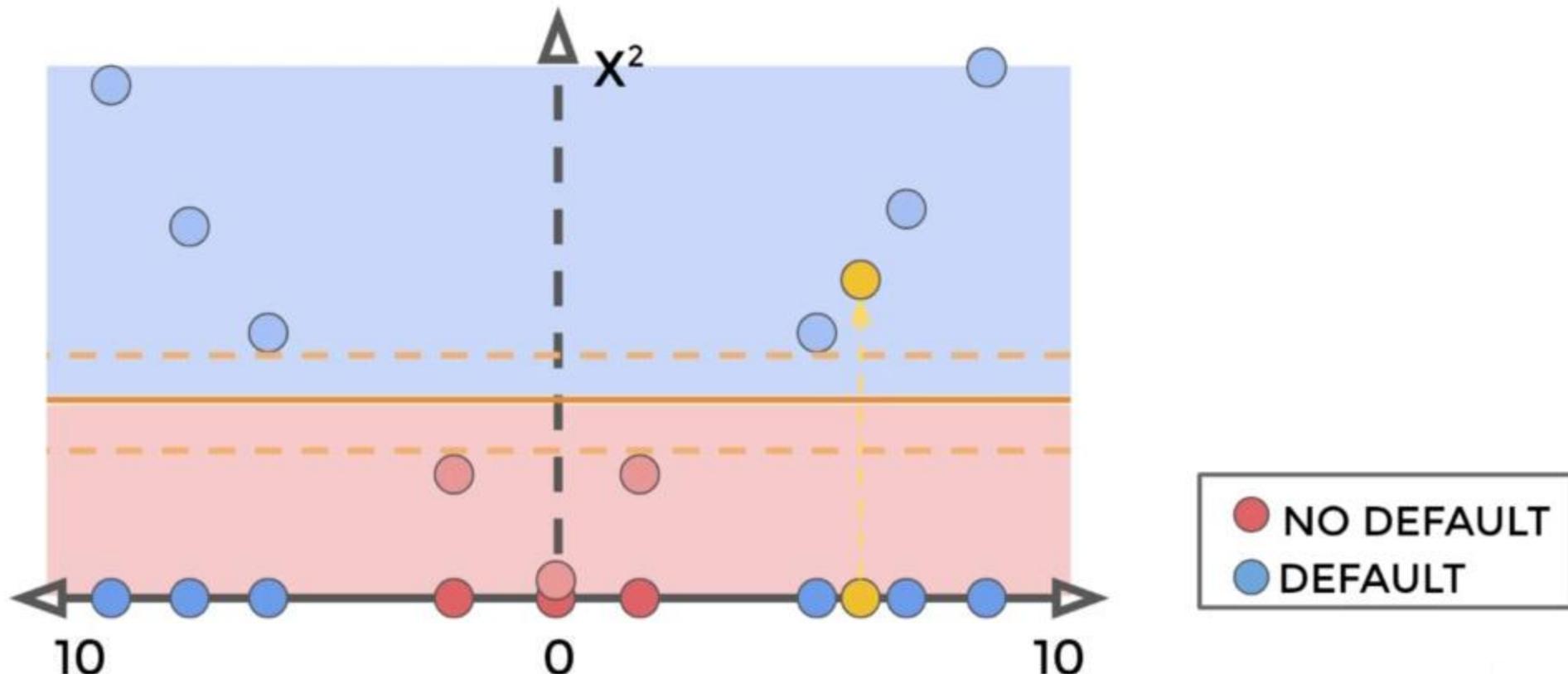
# Support Vector Machines

- Use kernel projection to evaluate new points:



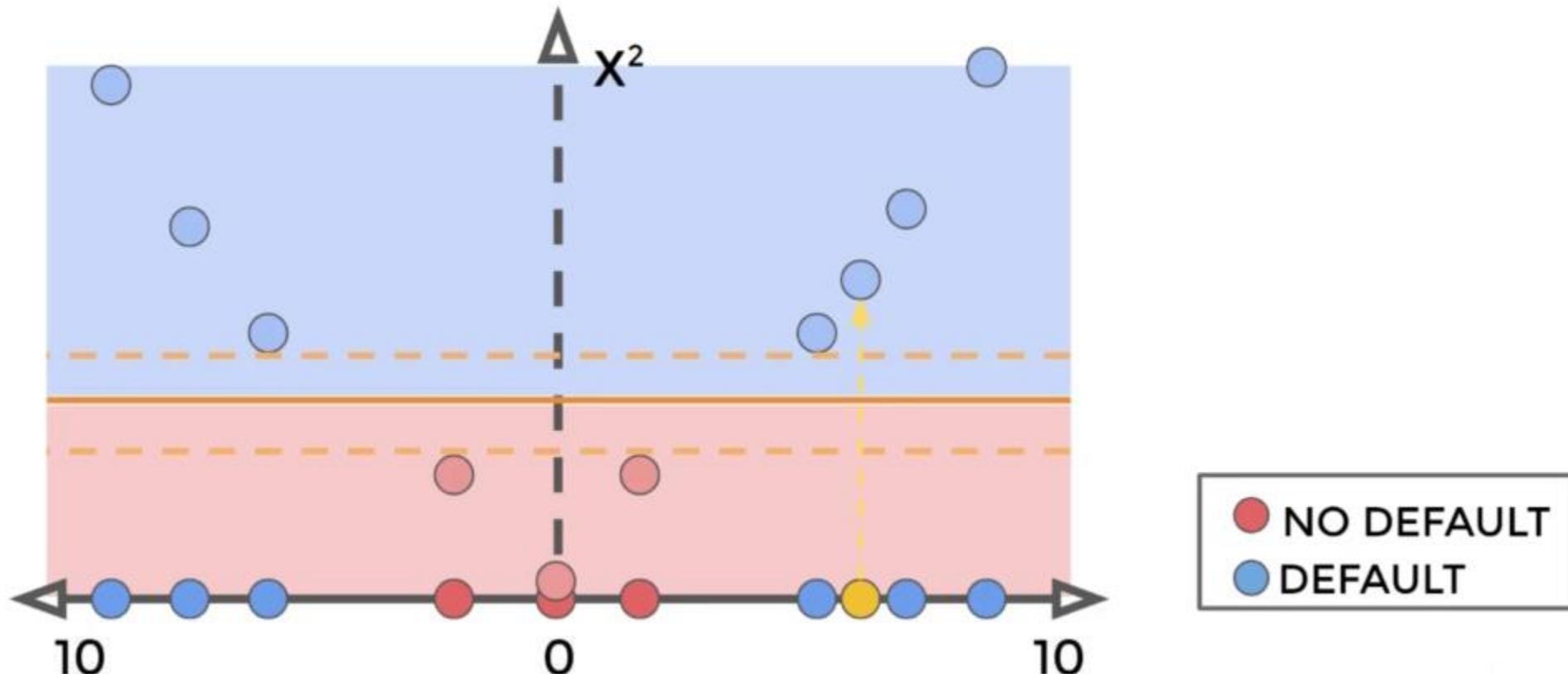
# Support Vector Machines

- Use kernel projection to evaluate new points:



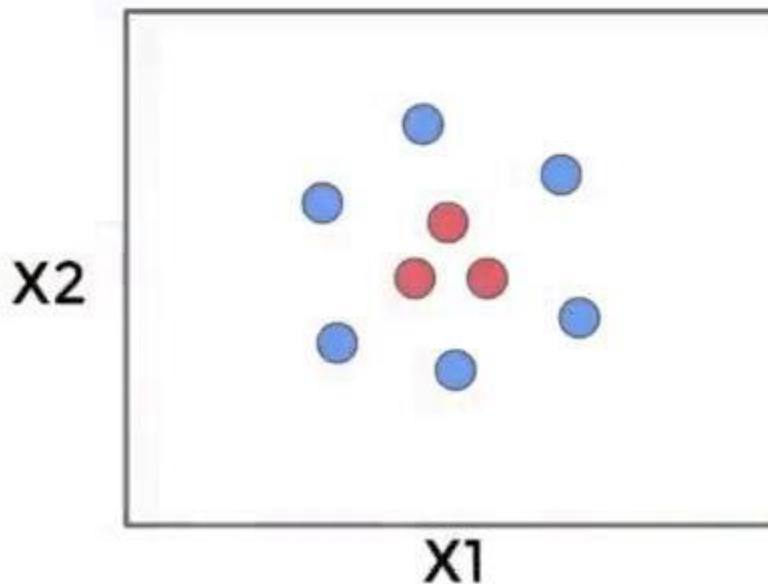
# Support Vector Machines

- Use kernel projection to evaluate new points:



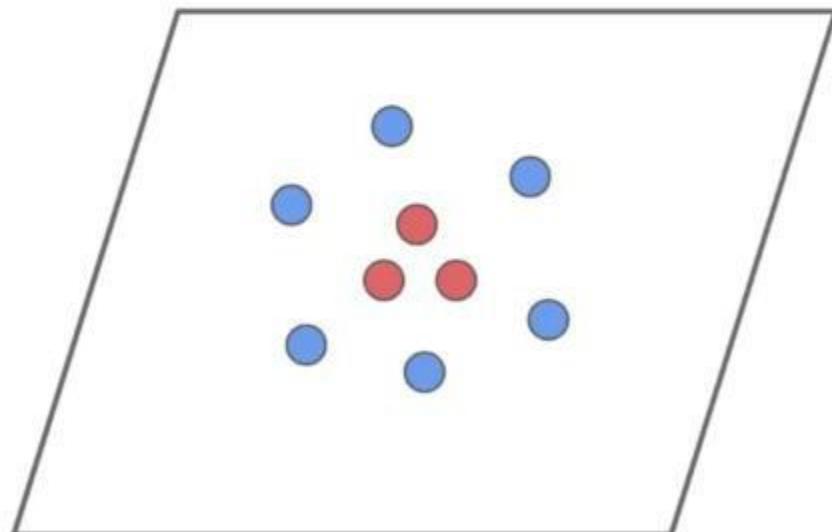
# Support Vector Machines

- Imagine a 2D feature space where a hyperplane can not separate effectively, even with soft margins.



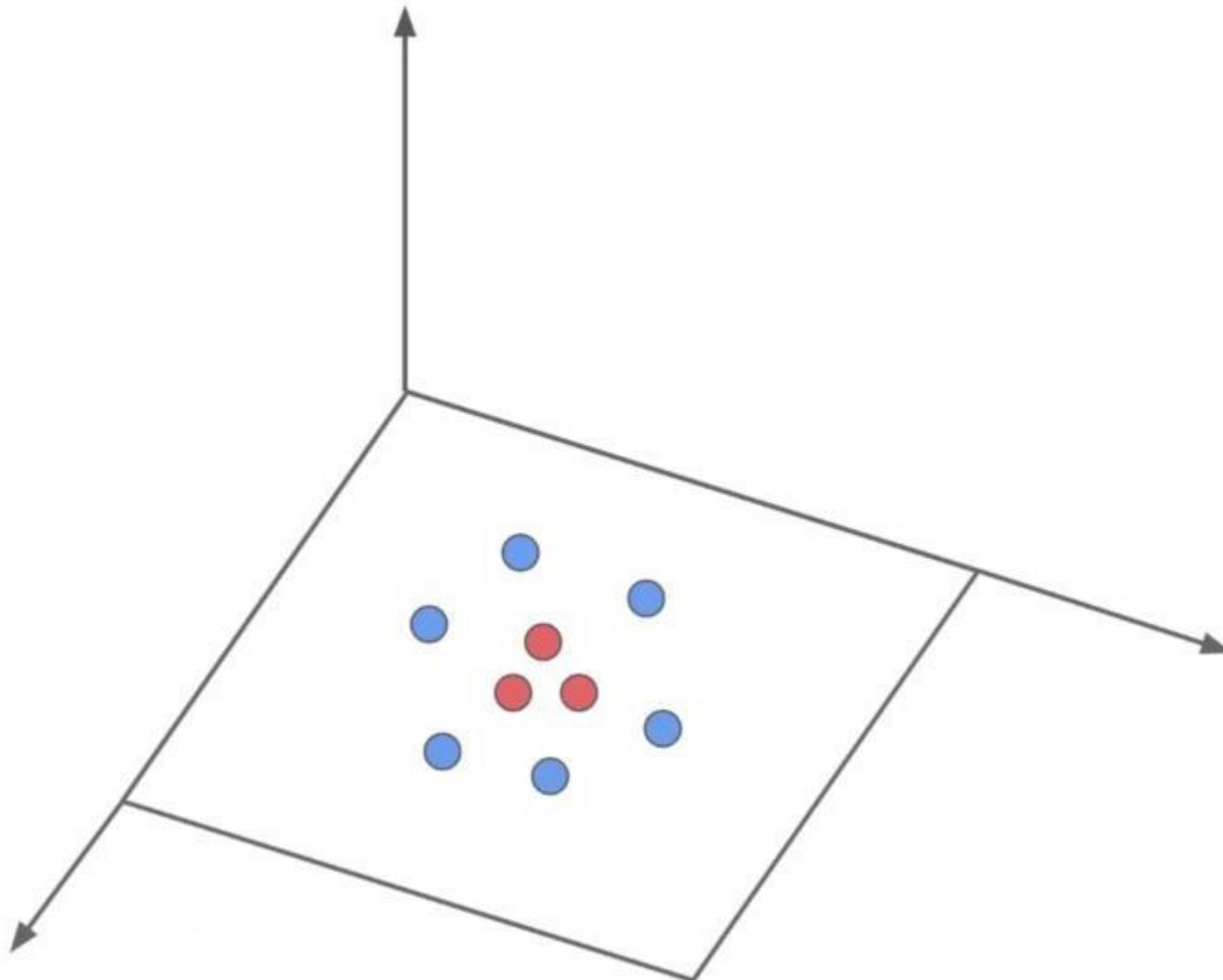
# Support Vector Machines

- We use Support Vector Machines to enable the use of a kernel transformation to project to a higher dimension.



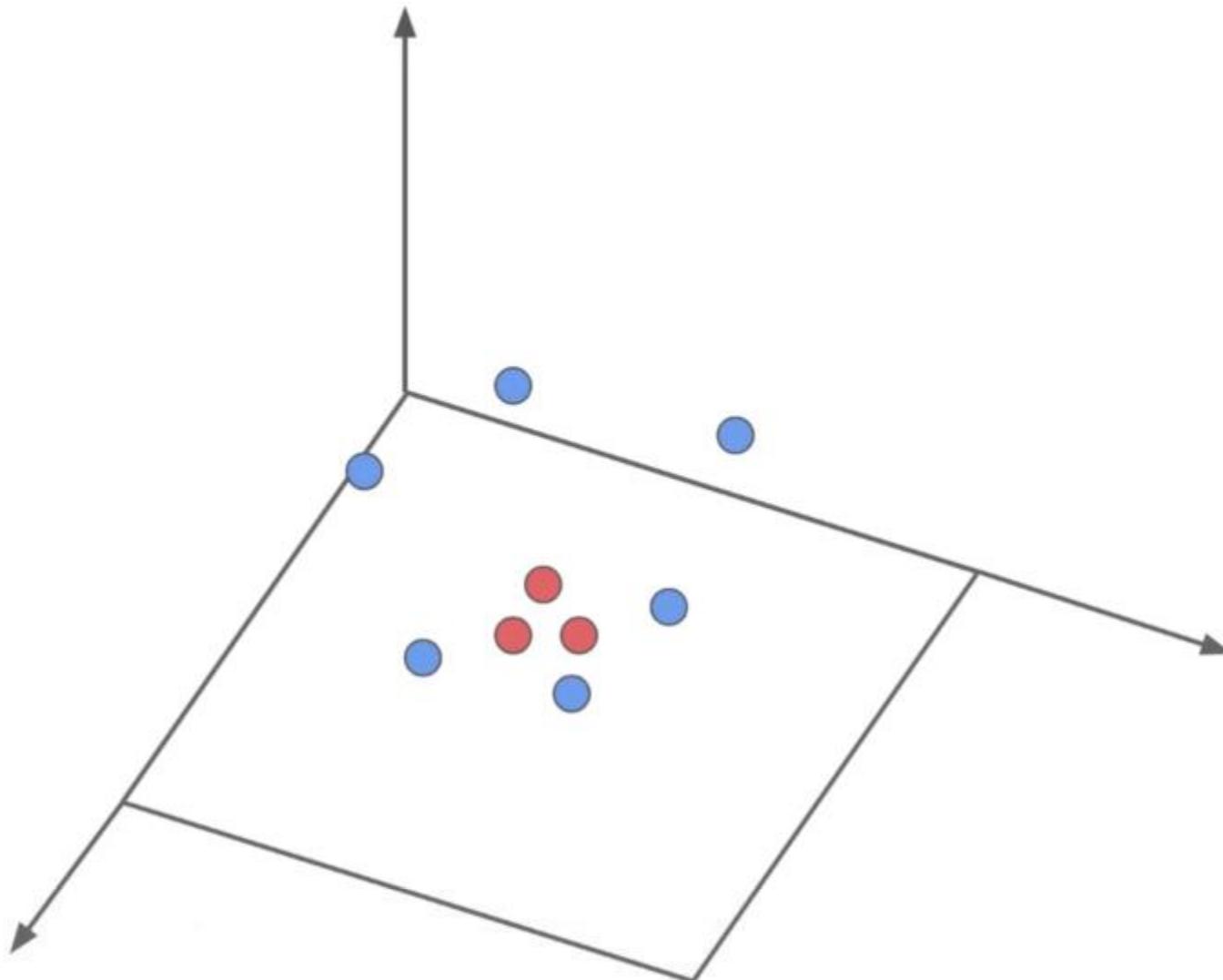
# Support Vector Machines

- 2D to 3D



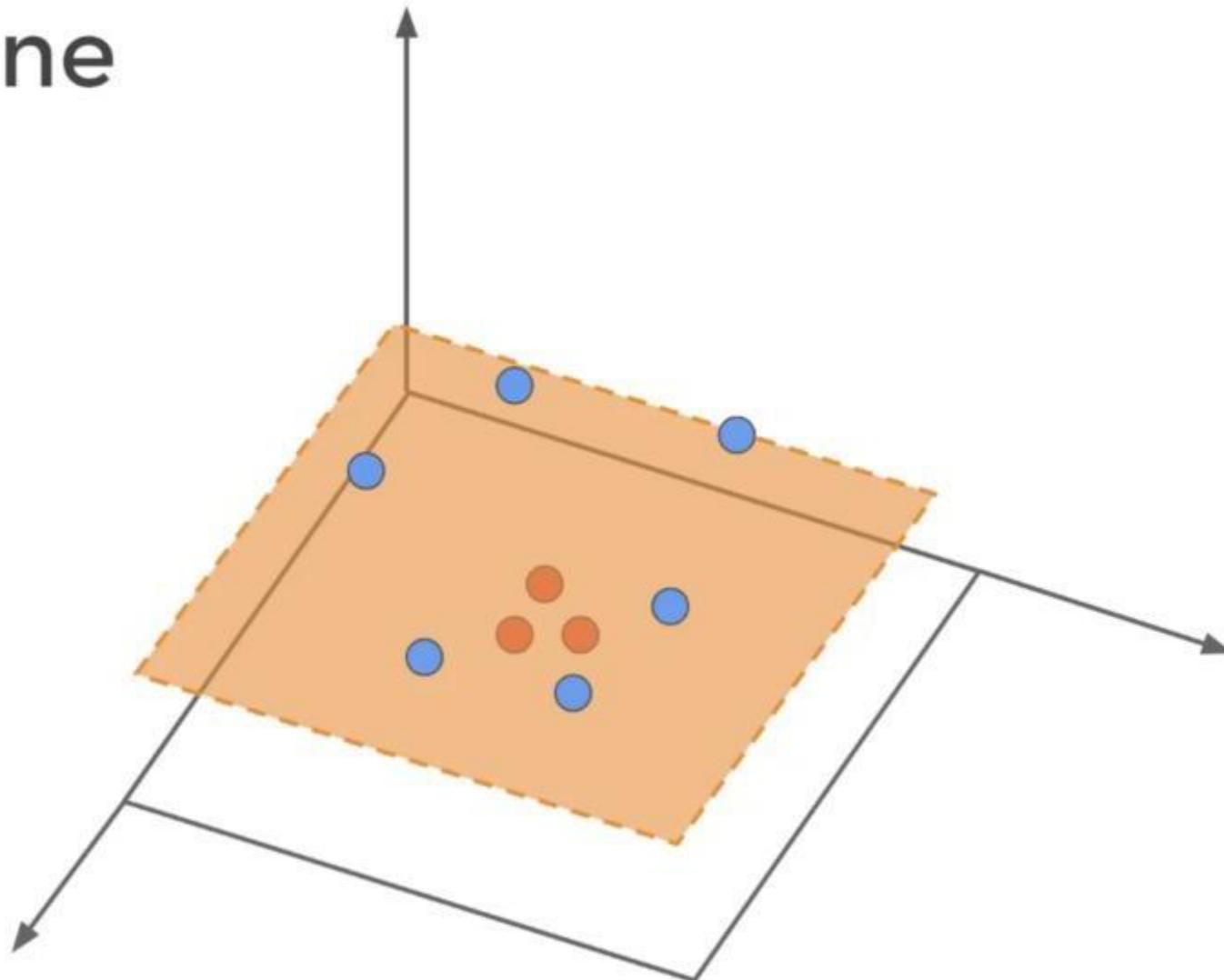
# Support Vector Machines

- 2D to 3D



# Support Vector Machines

- Hyperplane



# Support Vector Machines

- You may have heard of the use of kernels in SVM as the “**kernel trick**”.
- We previously visualized transforming data points from one dimension into a higher dimension.
- Mathematically, the **kernel trick** actually avoids recomputing the points in a higher dimensional space!

# Support Vector Machines

- How does the kernel trick achieve this?
- It takes advantage of dot products of the transpositions of the data.
- In the next lecture, we will go through the basic mathematical ideas behind the “kernel trick”!



# Support Vector Machines

Theory and Intuition - Kernel Trick and Math

# Support Vector Machines

- Let's briefly go over some of the general mathematics of SVM and how it is related to the Scikit-Learn class calls.
- We'll begin with a brief review of using margin based classifiers, and how they can be described with equations.
  - Note: *Feel free to consider this an “optional” lecture.*

# Support Vector Machines

- Relative Reading:
  - Background in Chapter 9 of ISLR.
  - For a comprehensive overview of everything discussed, check out:
    - Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks". *Machine Learning*.

# Support Vector Machines

- Hyperplanes Defined

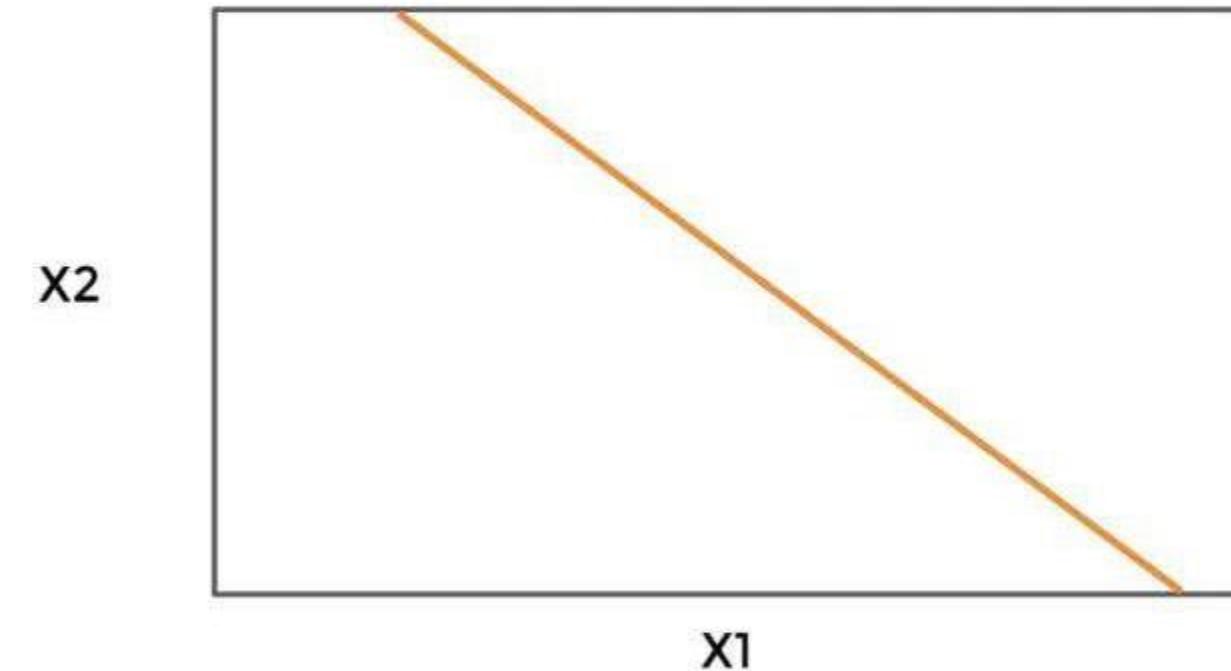


X1

X2

# Support Vector Machines

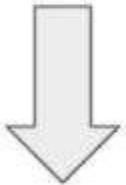
- Hyperplanes Defined



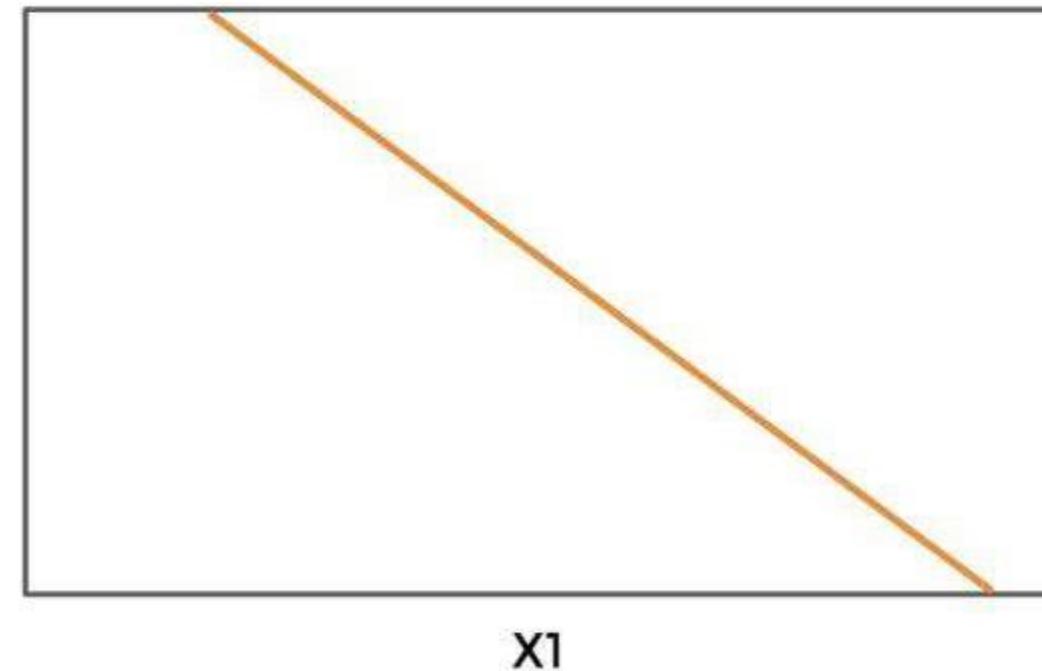
# Support Vector Machines

- Hyperplanes Defined

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

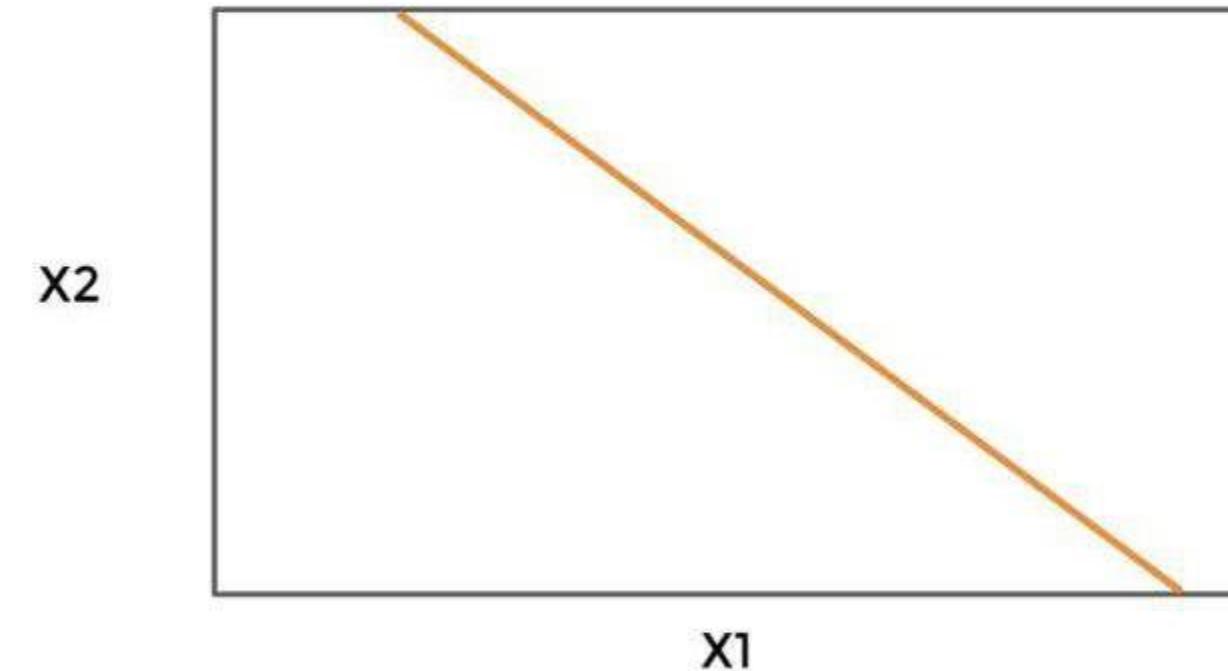


$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p = 0$$



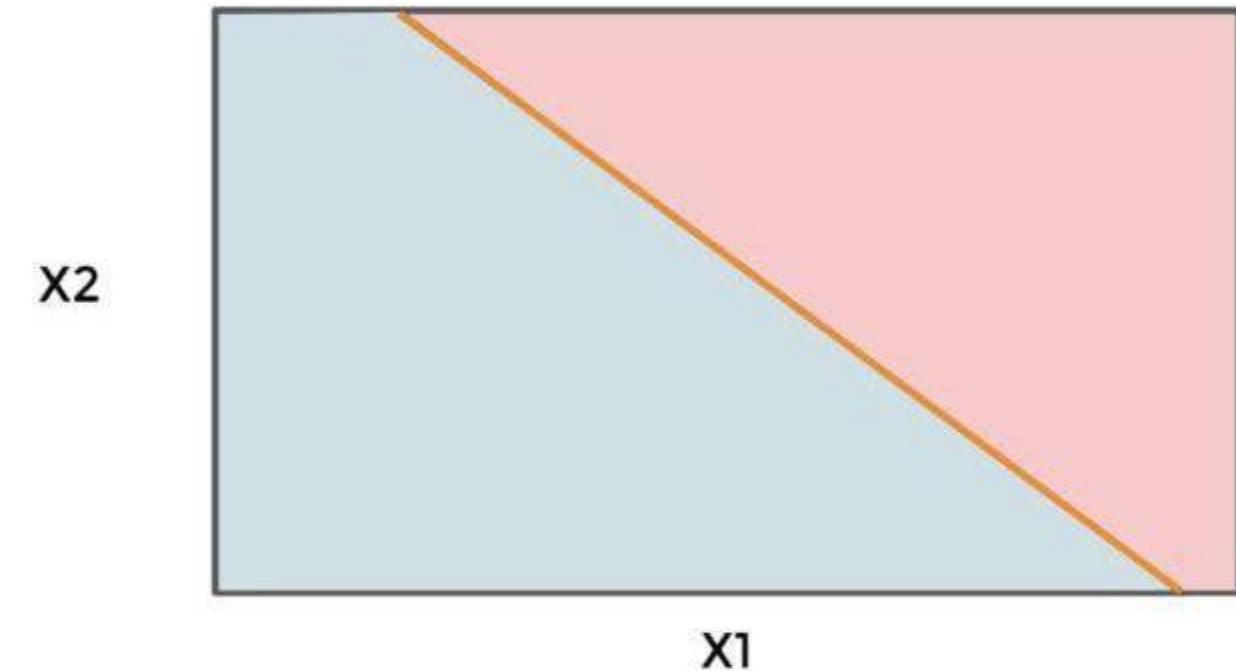
# Support Vector Machines

- Separating Hyperplanes



# Support Vector Machines

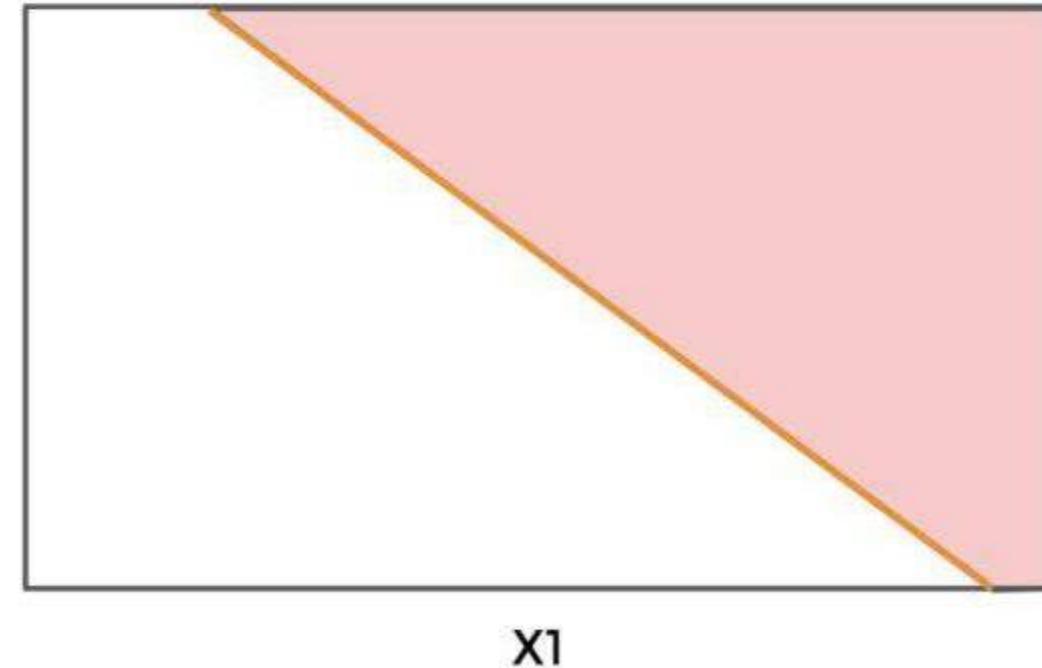
- Separating Hyperplanes



# Support Vector Machines

- Separating Hyperplanes

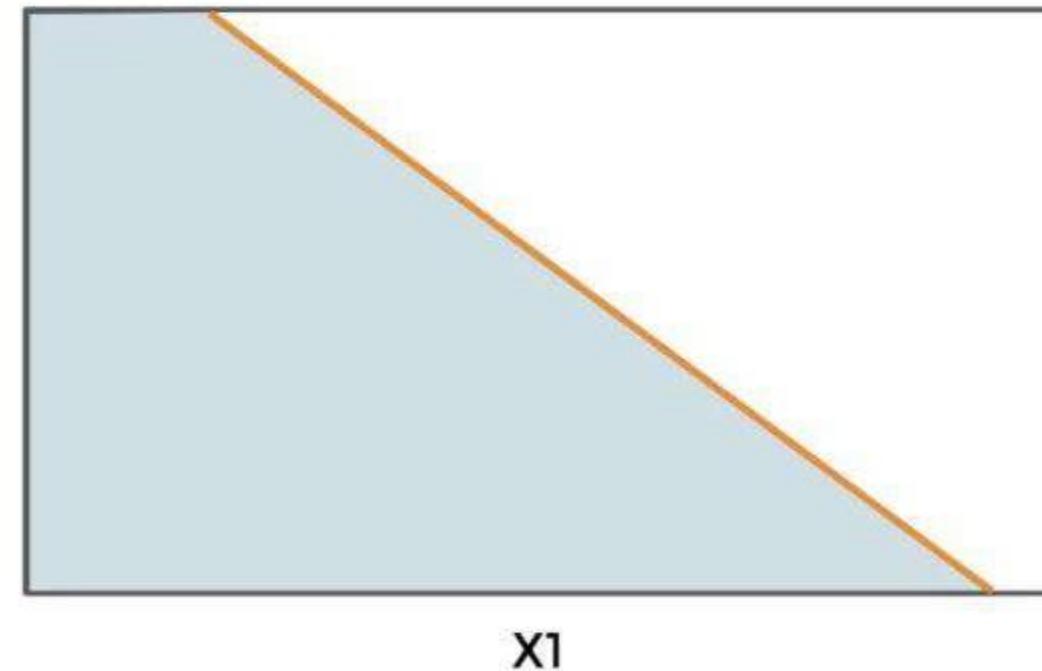
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p > 0$$



# Support Vector Machines

- Separating Hyperplanes

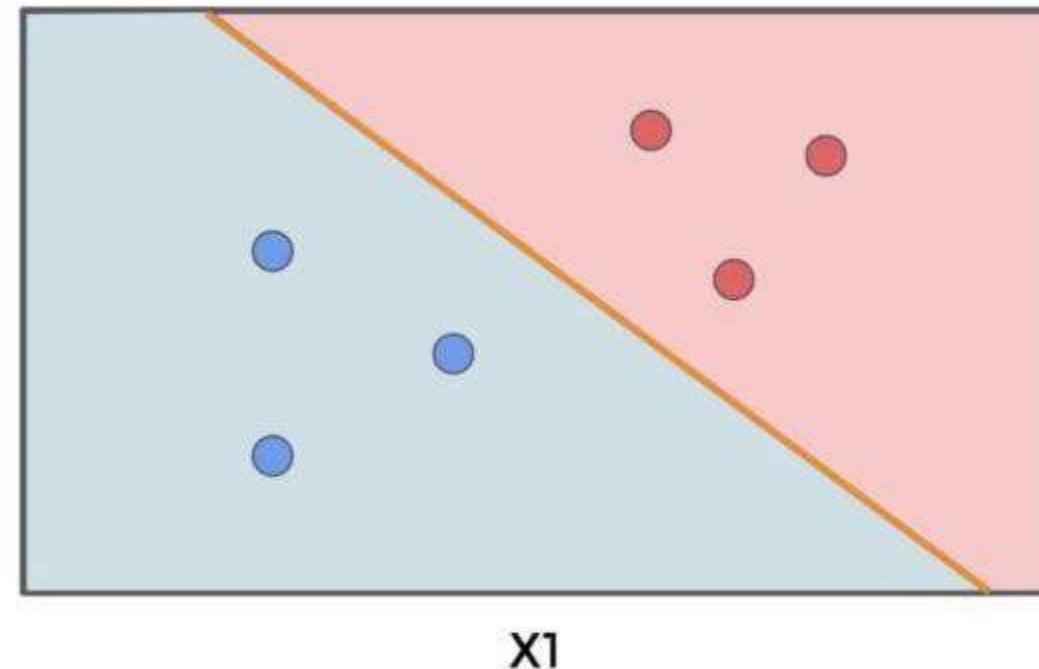
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p < 0$$



# Support Vector Machines

- Data Points

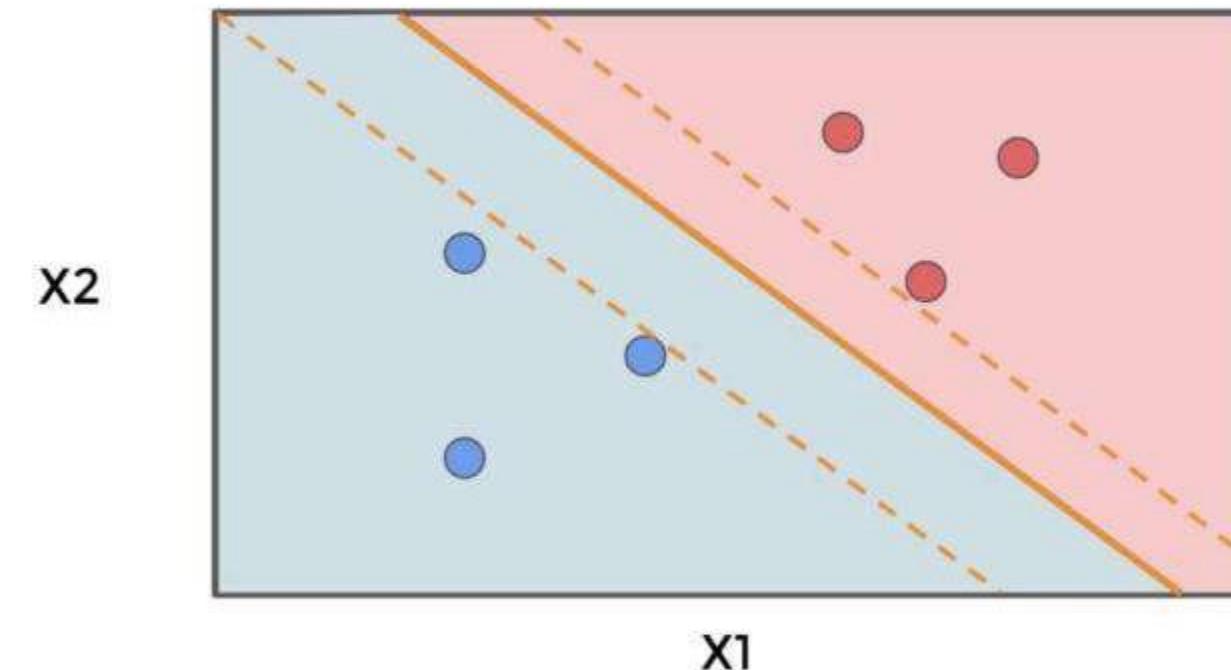
$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$



# Support Vector Machines

- Max Margin Classifier

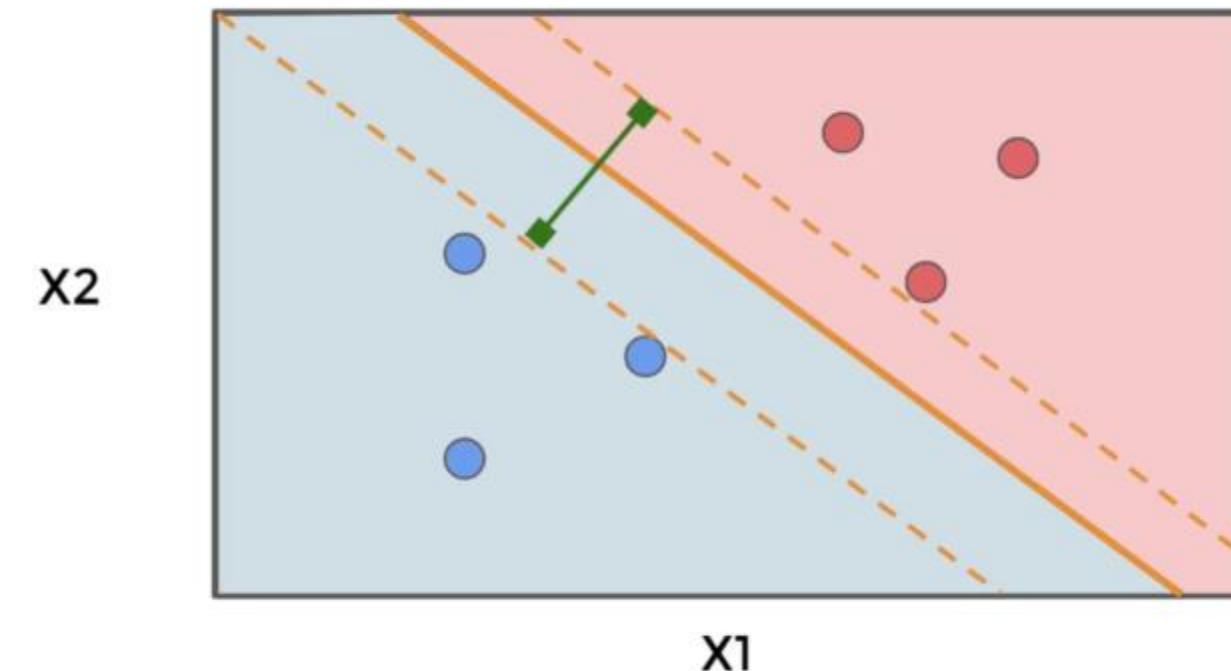
$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$



# Support Vector Machines

- Max Margin Classifier

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \boxed{M}$$



# Support Vector Machines

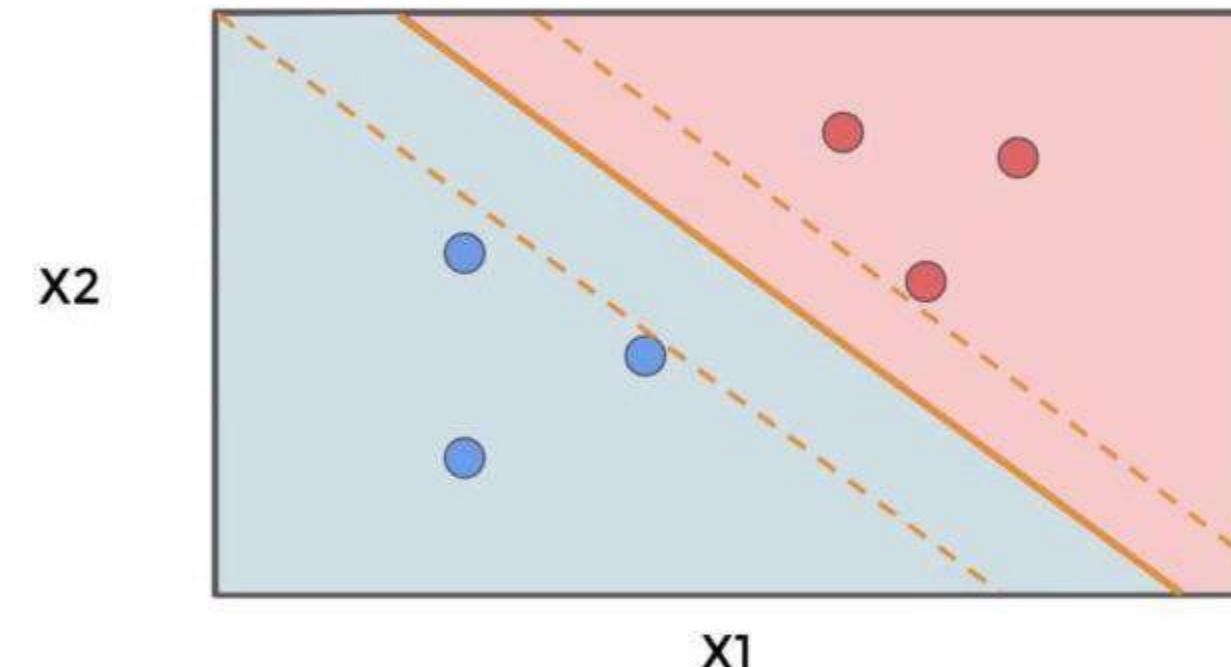
- Max Margin Classifier

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^p \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n.$$



# Support Vector Machines

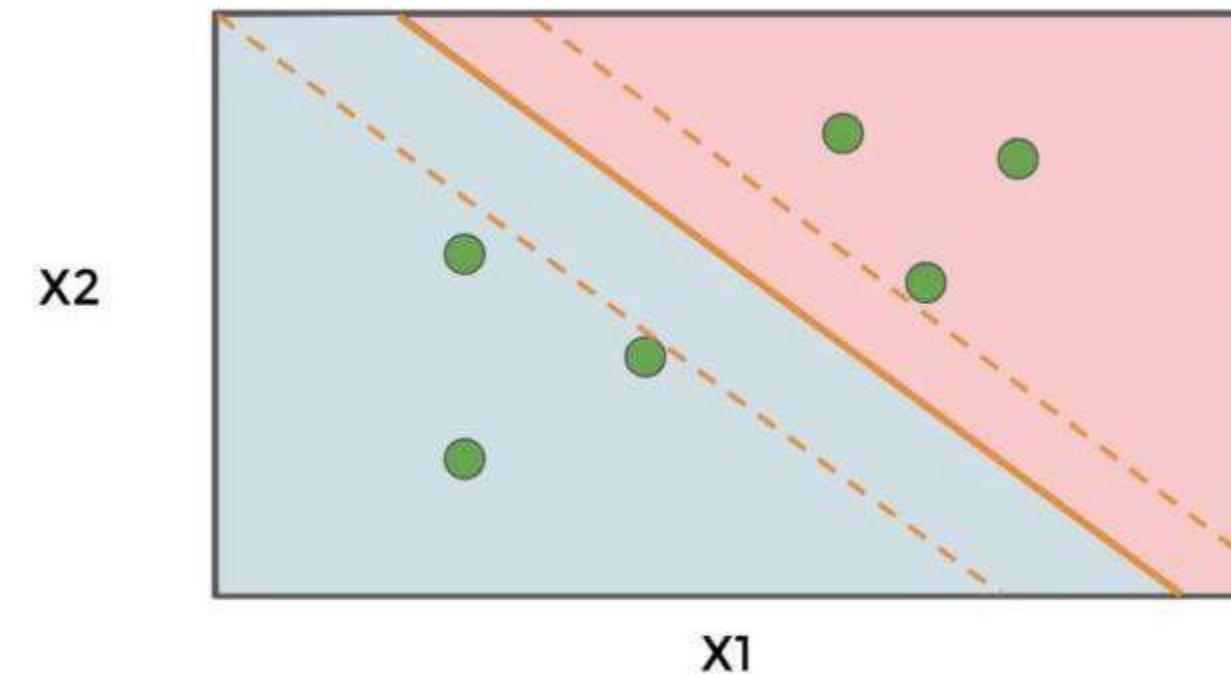
- Max Margin Classifier

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^p \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n.$$



# Support Vector Machines

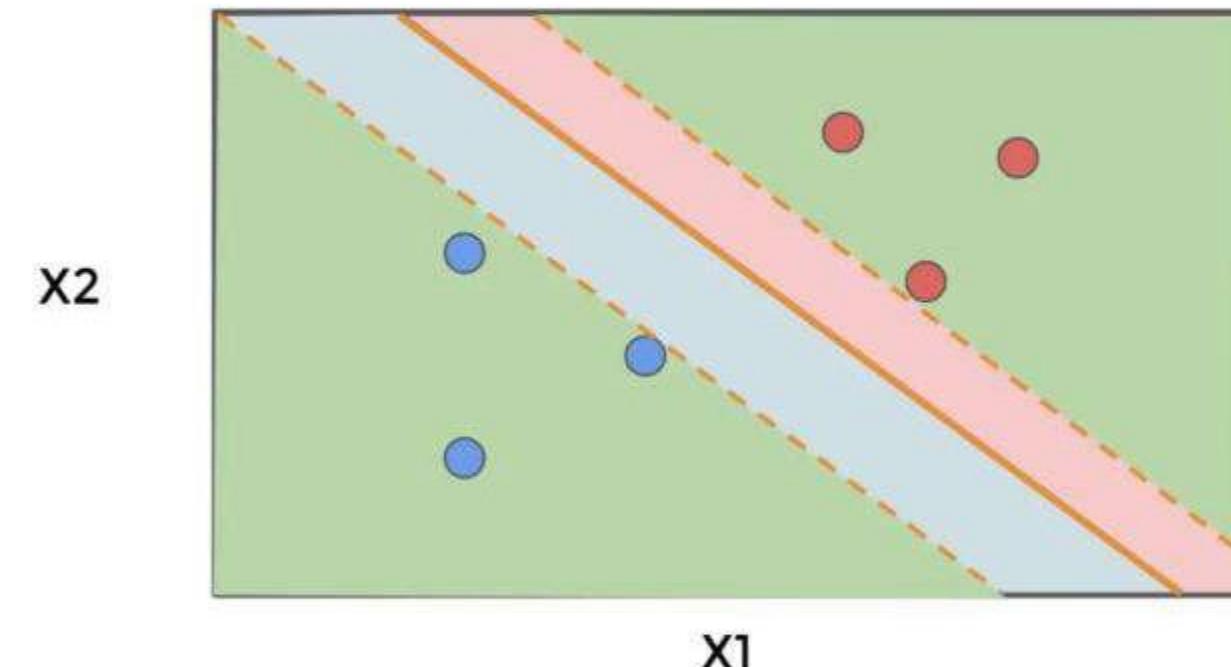
- Max Margin Classifier

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^p \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n.$$



# Support Vector Machines

- Max Margin Classifier

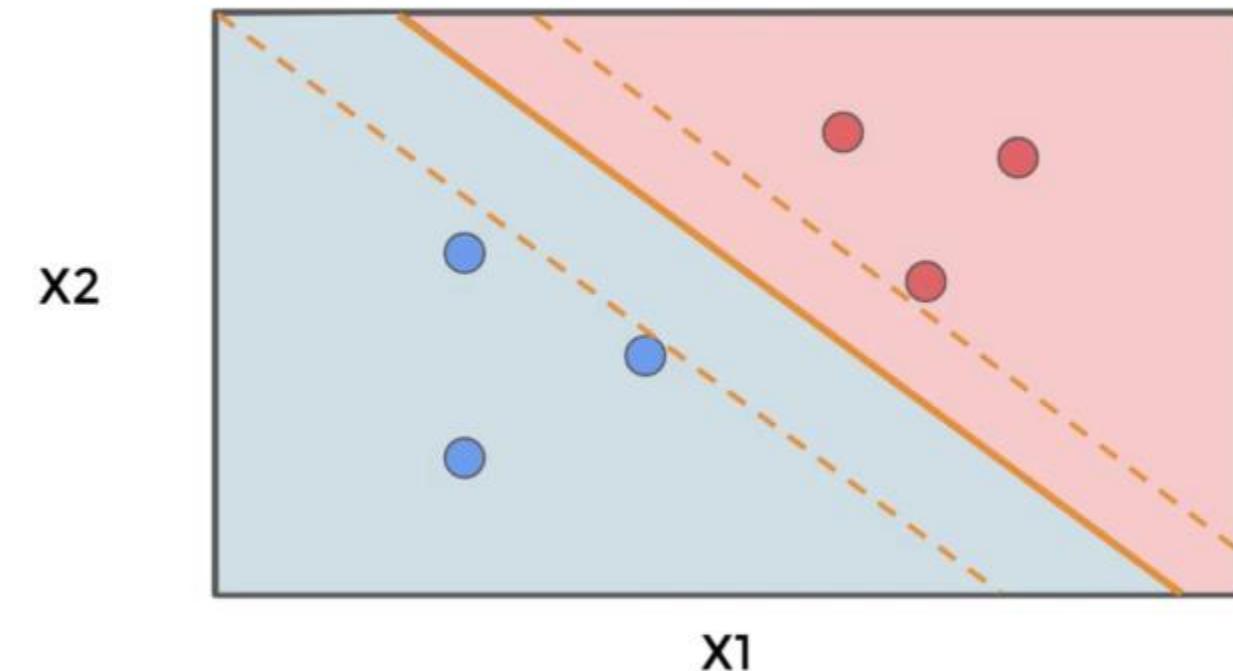
$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$

subject to

$$\sum_{j=1}^p \beta_j^2 = 1$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n.$$

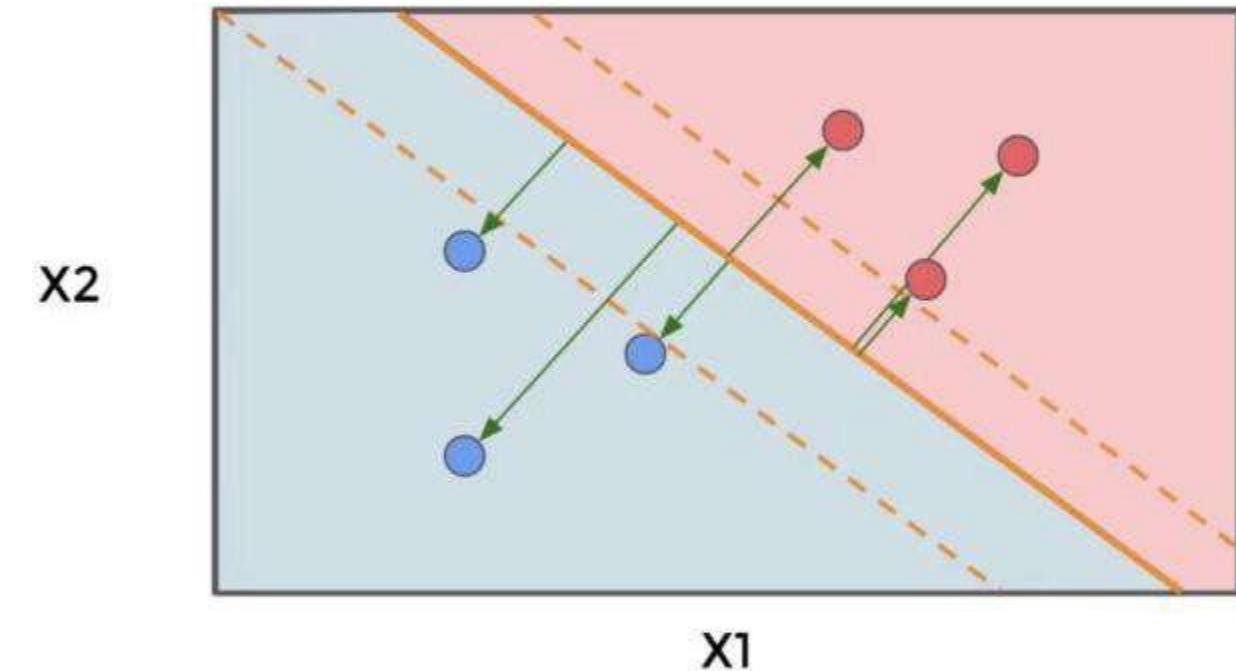


# Support Vector Machines

- Max Margin Classifier

subject to  $\sum_{j=1}^p \beta_j^2 = 1$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip})$$



# Support Vector Machines

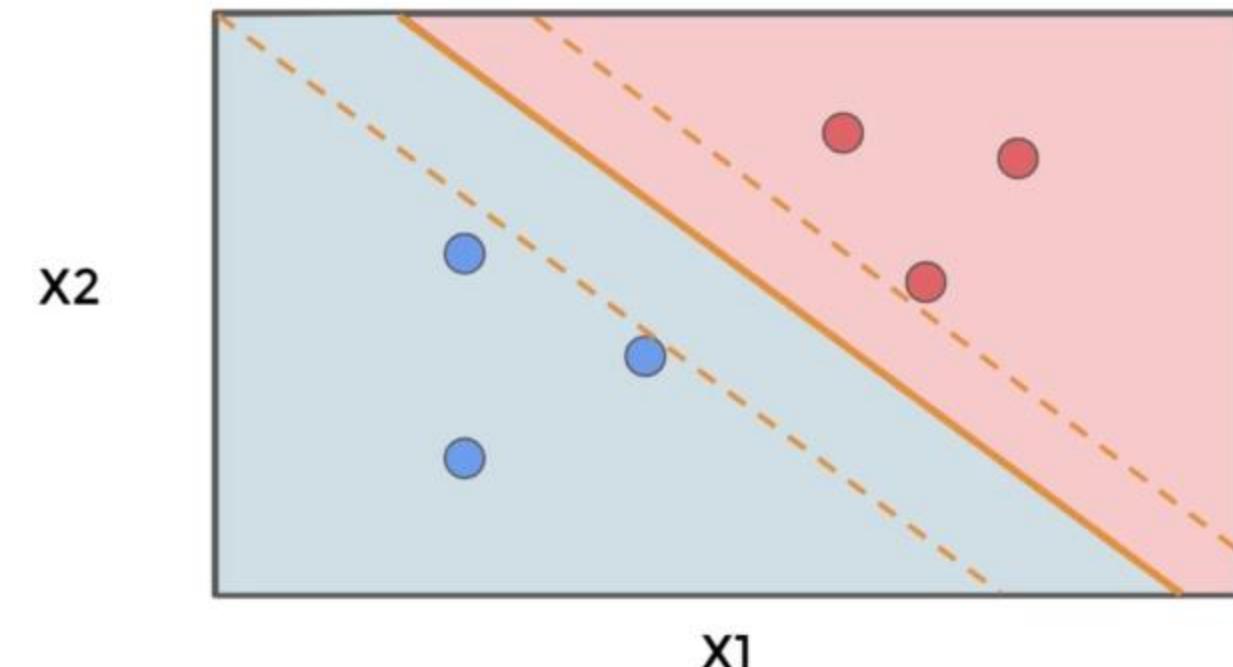
- Max Margin Classifier

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

$$\underset{\beta_0, \beta_1, \dots, \beta_p, M}{\text{maximize}} \quad M$$

$$\text{subject to} \quad \sum_{j=1}^p \beta_j^2 = 1$$

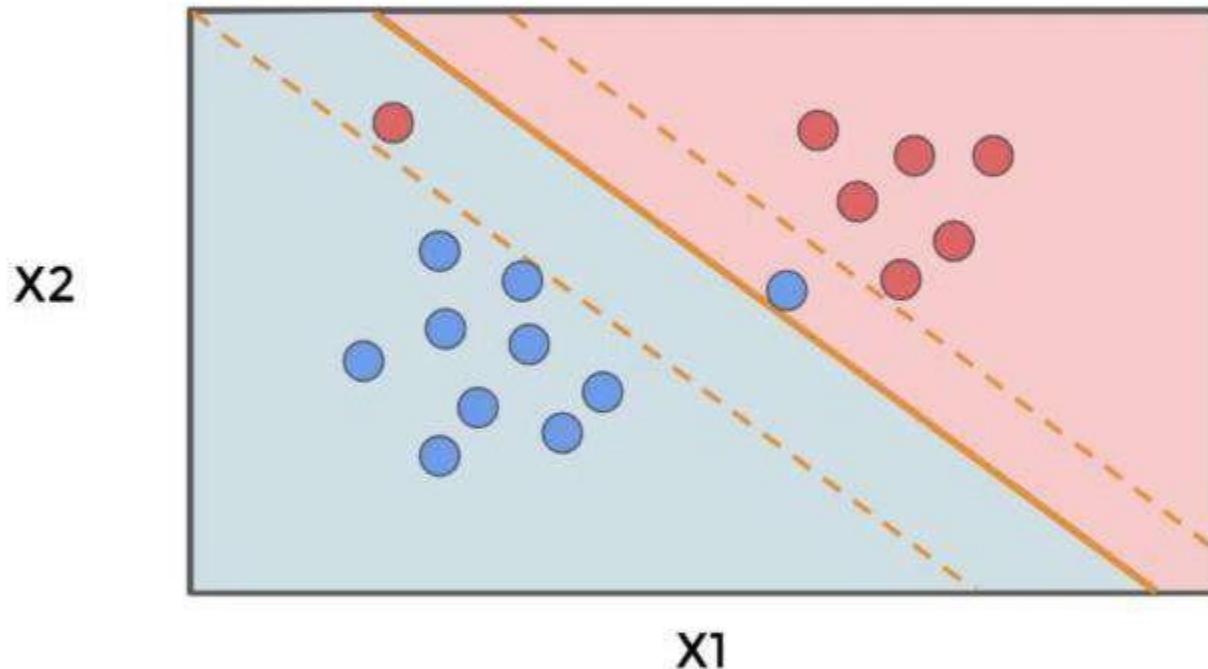
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M \quad \forall i = 1, \dots, n.$$



# Support Vector Machines

- Support Vector Classifier

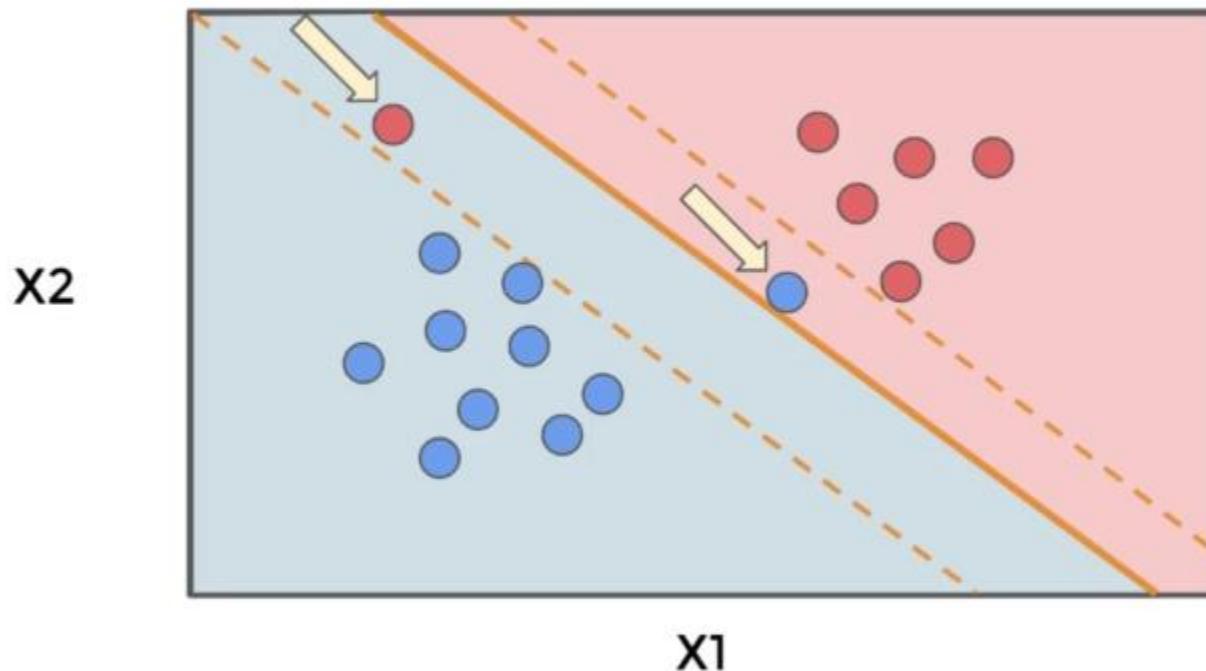
$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$



# Support Vector Machines

- Support Vector Classifier

$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \dots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$



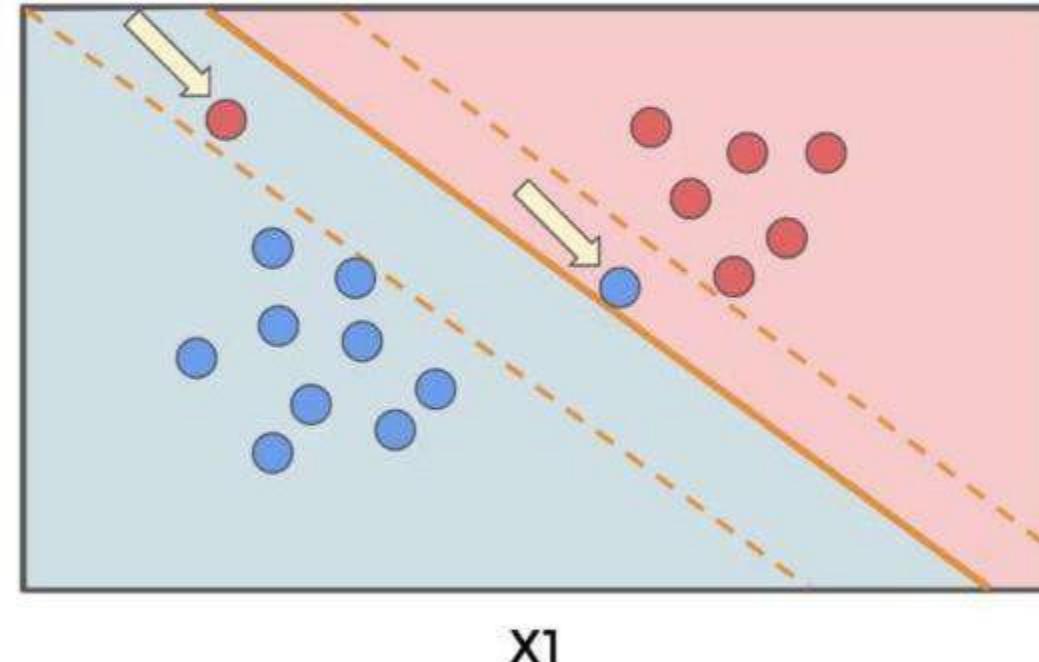
# Support Vector Machines

- Support Vector Classifier

$$\underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} \quad M$$

subject to  $\sum_{j=1}^p \beta_j^2 = 1$

$x_2$



# Support Vector Machines

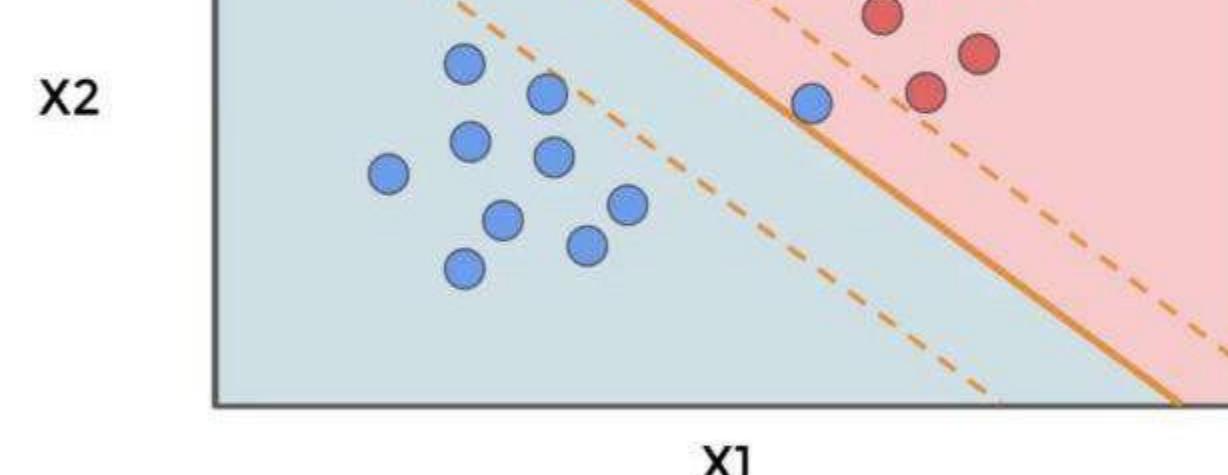
- Support Vector Classifier

$$\underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} \quad M$$

subject to  $\sum_{j=1}^p \beta_j^2 = 1$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^n \epsilon_i \leq C$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$



# Support Vector Machines

- Support Vector Classifier

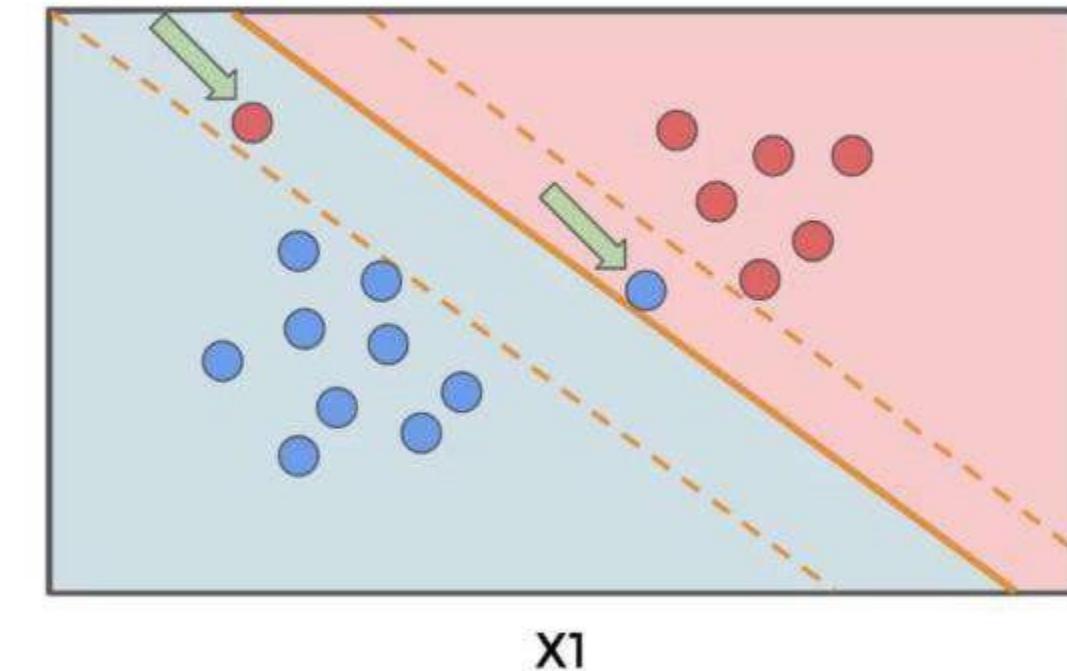
$$\underset{\beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} \quad M$$

subject to  $\sum_{j=1}^p \beta_j^2 = 1$

$$\epsilon_i \geq 0, \quad \sum_{i=1}^n \epsilon_i \leq C$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$

$x_2$



$X1$

# Support Vector Machines

- Note on Scikit-Learn's SVC!

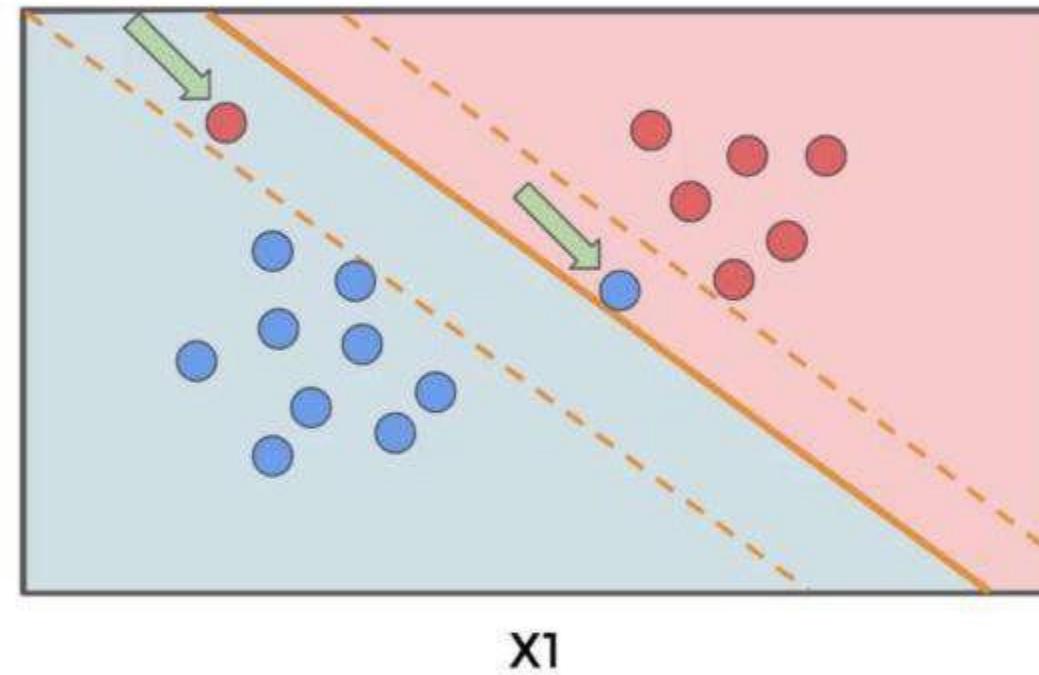
**C : float, default=1.0**

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive. The penalty is a squared L2 penalty.

$$\epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C$$

$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i)$$

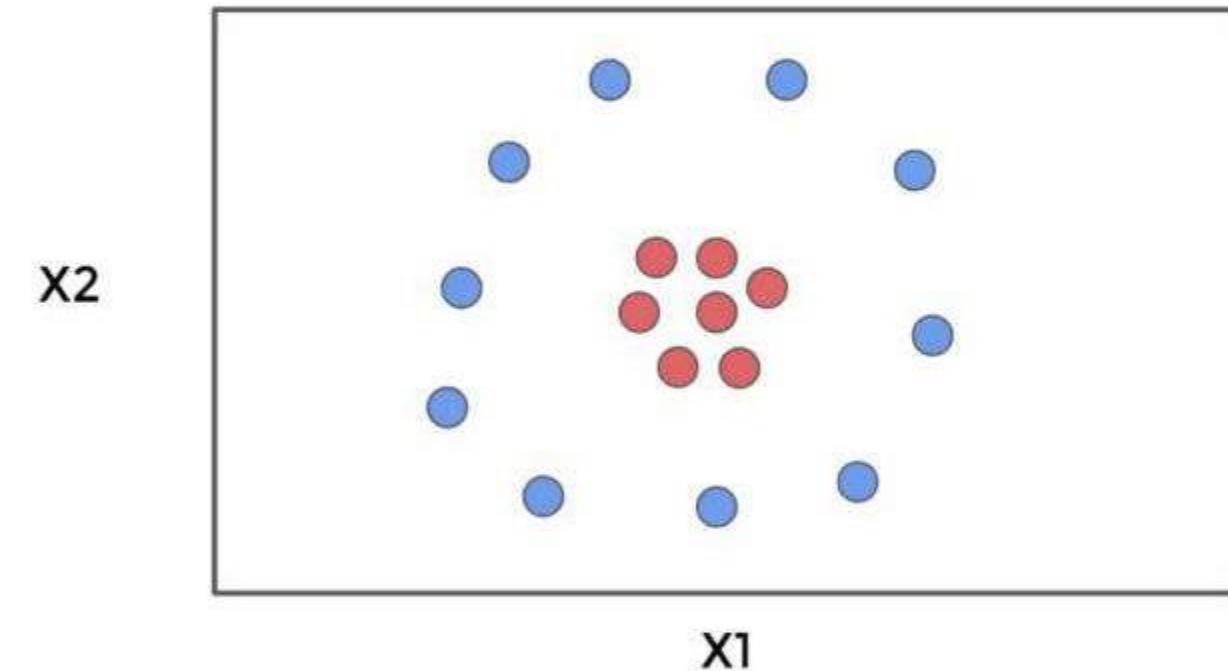
x2



# Support Vector Machines

- Support Vector Machines

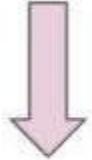
$X_1, X_2, \dots, X_p,$



# Support Vector Machines

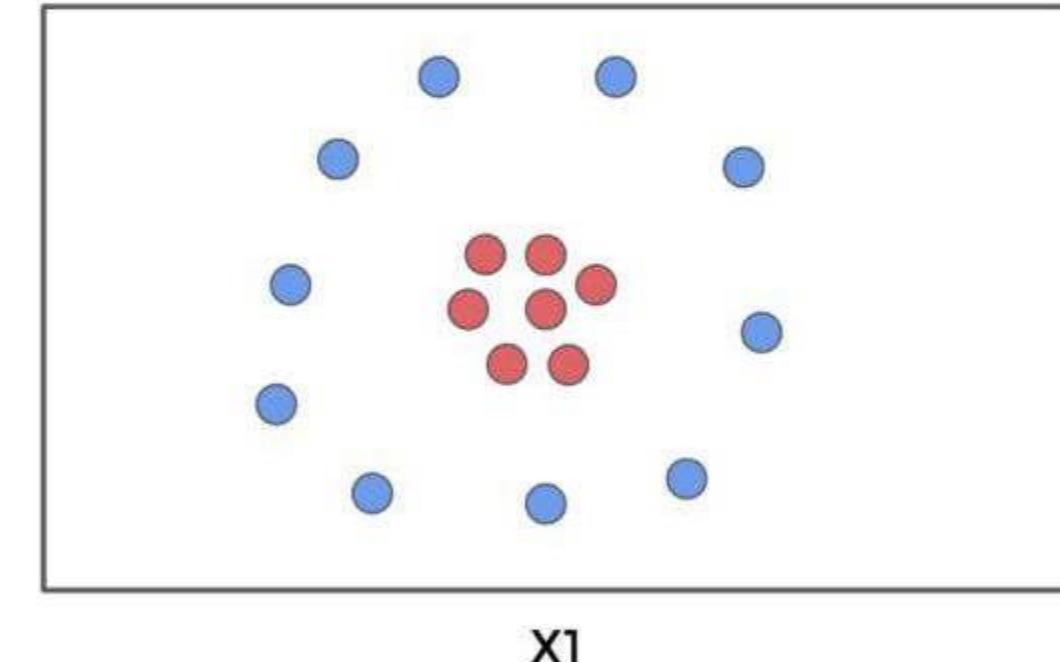
- Support Vector Machines

$X_1, X_2, \dots, X_p,$



$X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$

$x_2$



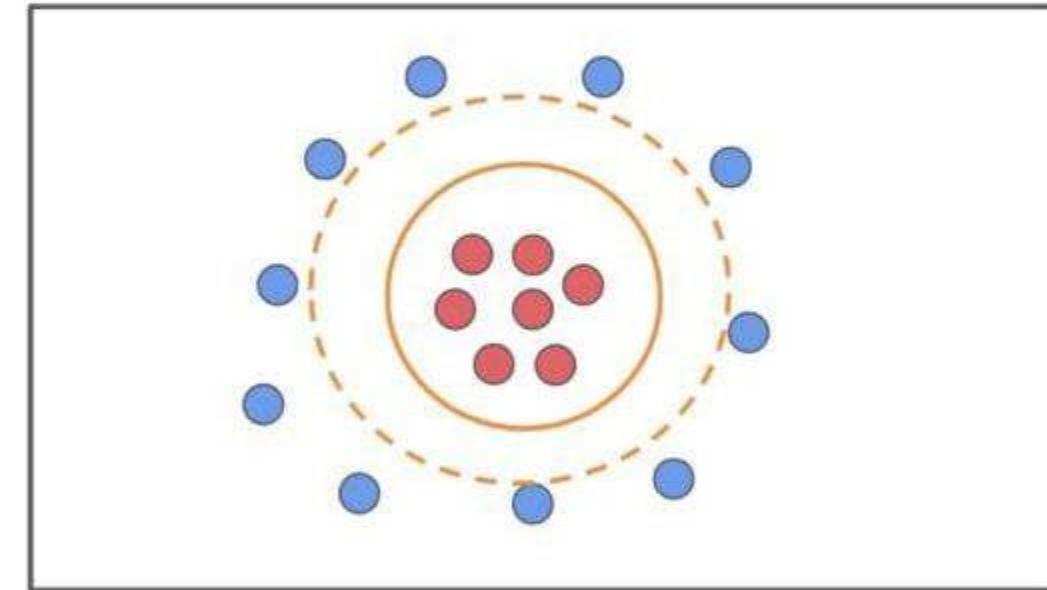
$x_1$

# Support Vector Machines

- Support Vector Machines

$$\underset{\beta_0, \beta_{11}, \beta_{12}, \dots, \beta_{p1}, \beta_{p2}, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} \quad M$$

X2



X1

# Support Vector Machines

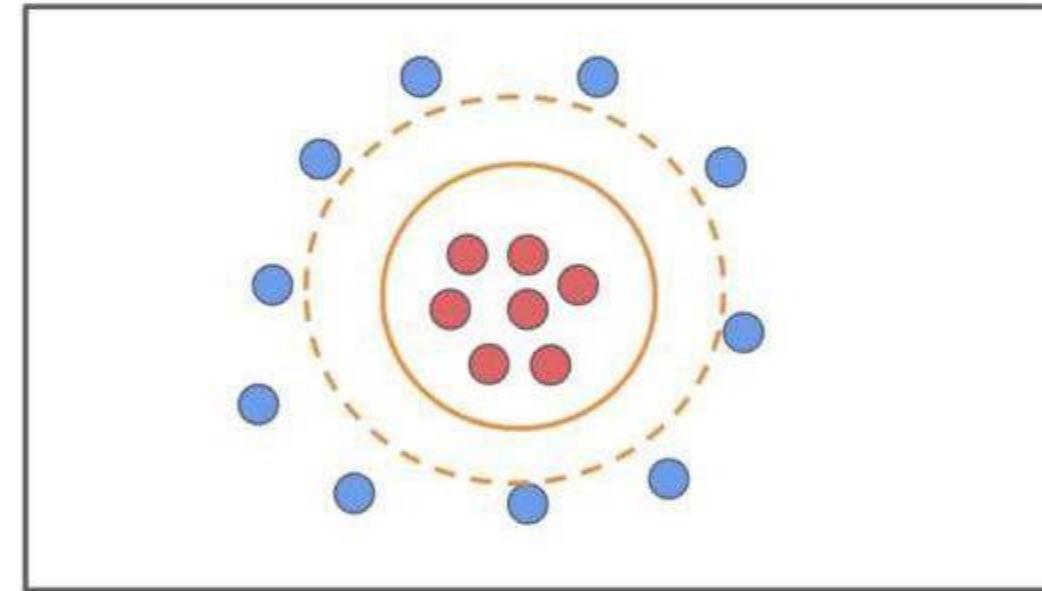
- Support Vector Machines

$$X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$$

$$\underset{\beta_0, \beta_{11}, \beta_{12}, \dots, \beta_{p1}, \beta_{p2}, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}}$$

subject to  $y_i \left( \beta_0 + \sum_{j=1}^p \beta_{j1} x_{ij} + \sum_{j=1}^p \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i)$

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^p \sum_{k=1}^2 \beta_{jk}^2 = 1.$$



# Support Vector Machines

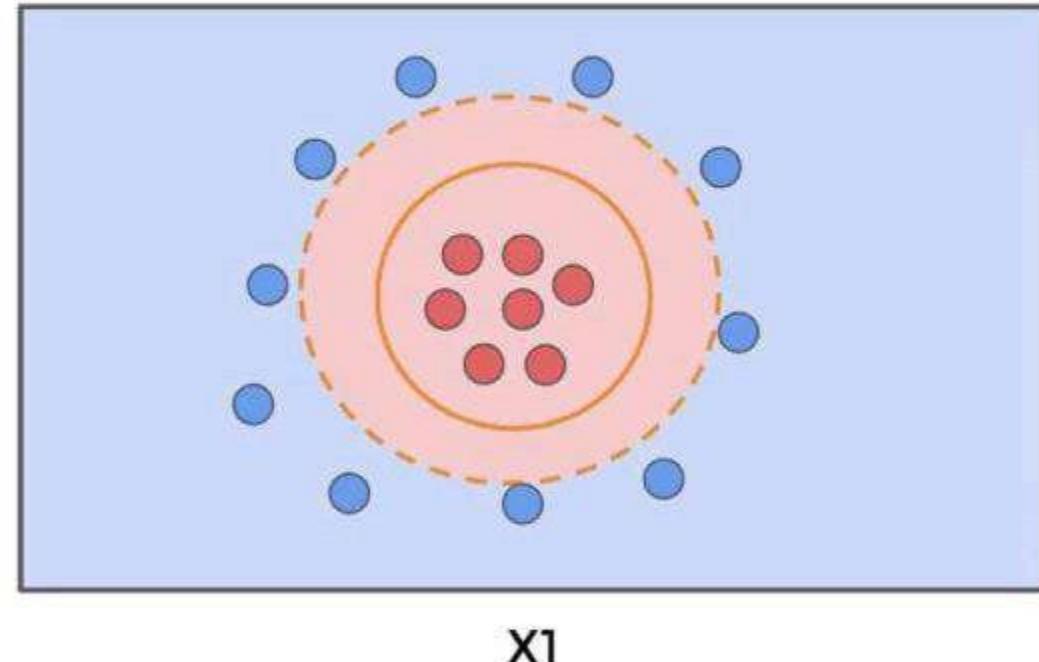
- Support Vector Machines

$$X_1, X_1^2, X_2, X_2^2, \dots, X_p, X_p^2$$

$$\underset{\beta_0, \beta_{11}, \beta_{12}, \dots, \beta_{p1}, \beta_{p2}, \epsilon_1, \dots, \epsilon_n, M}{\text{maximize}} \quad M$$

subject to  $y_i \left( \beta_0 + \sum_{j=1}^p \beta_{j1} x_{ij} + \sum_{j=1}^p \beta_{j2} x_{ij}^2 \right) \geq M(1 - \epsilon_i)$

$$\sum_{i=1}^n \epsilon_i \leq C, \quad \epsilon_i \geq 0, \quad \sum_{j=1}^p \sum_{k=1}^2 \beta_{jk}^2 = 1.$$



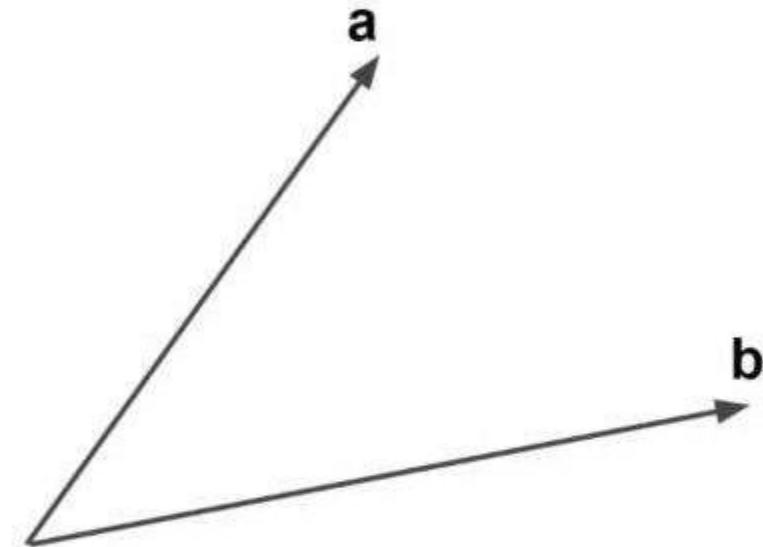
# Support Vector Machines

- How to deal with very large feature space?
- As polynomial order grows larger, the number of computations necessary to solve for margins also grows!
- The answer lies in the **kernel trick** which makes use of the **inner product** of vectors, also known as the **dot product**.

# Support Vector Machines

- Dot Product

$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$

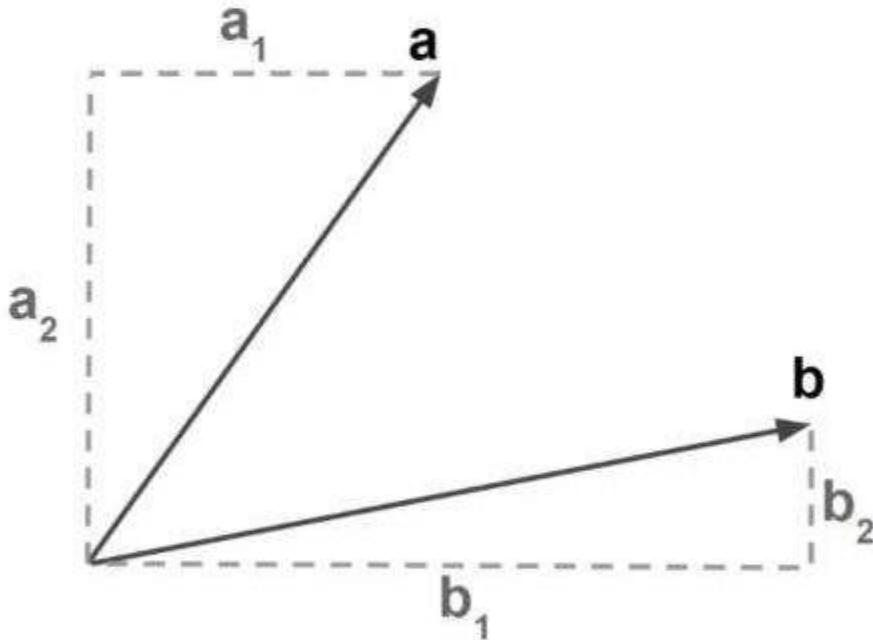


# Support Vector Machines

- Dot Product

$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$

$$a \cdot b = a_1 b_1 + a_2 b_2$$



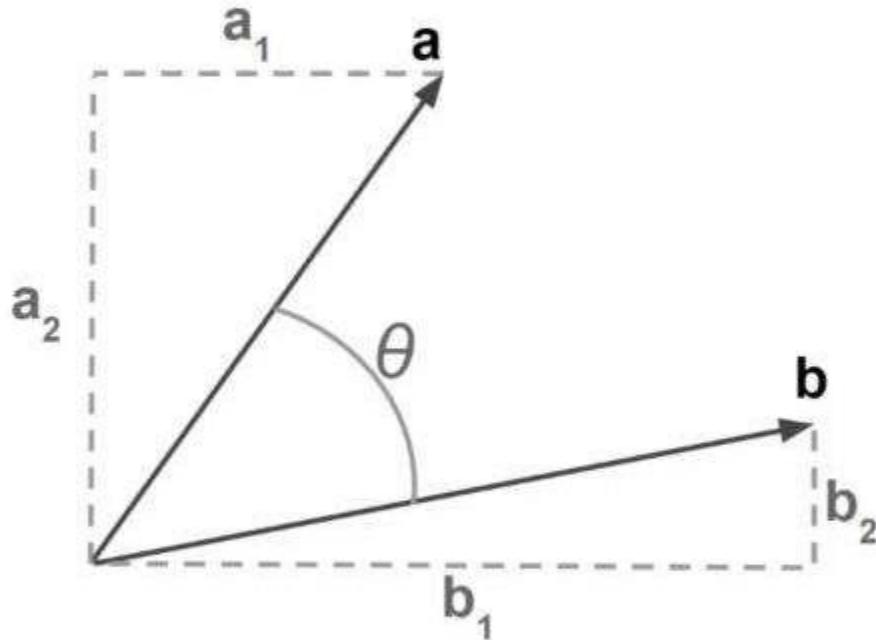
# Support Vector Machines

- Dot Product

$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$

$$a \cdot b = a_1 b_1 + a_2 b_2$$

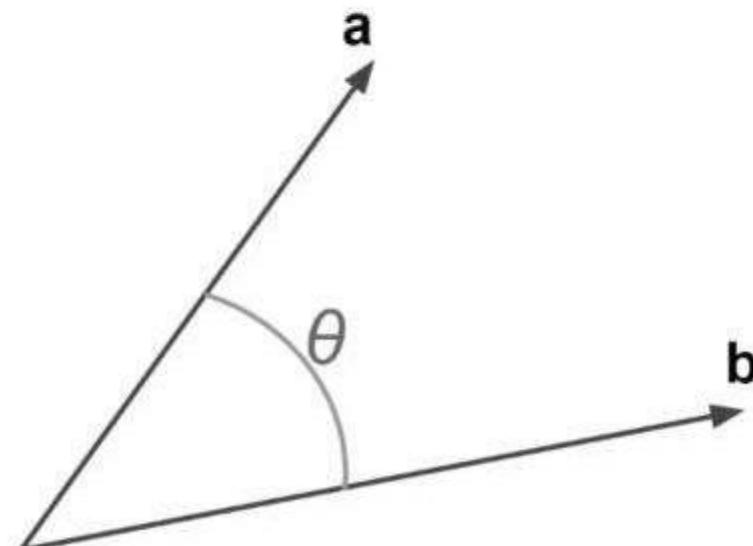
$$a \cdot b = |a| |b| \cos(\theta)$$



# Support Vector Machines

- Notice how the dot product can be thought of as a **similarity** between the vectors.

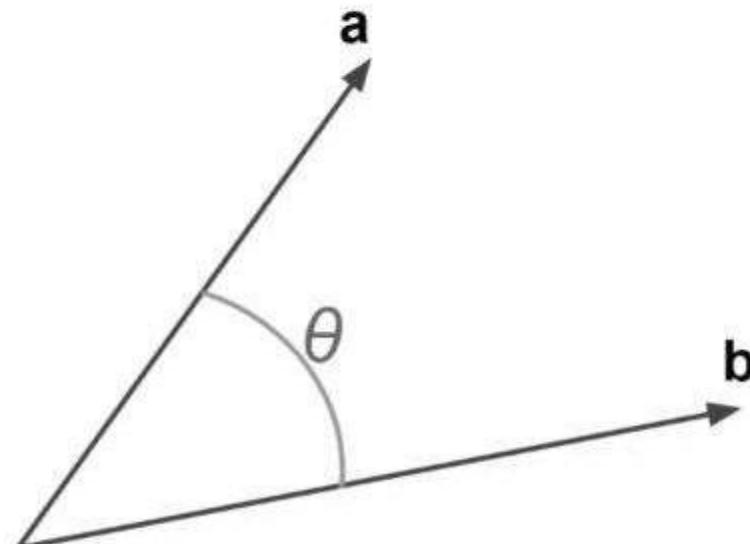
$$a \cdot b = |a||b|\cos(\theta)$$



# Support Vector Machines

- $\cos(0^\circ) = 1$
- $\cos(90^\circ) = 0$
- $\cos(180^\circ) = -1$

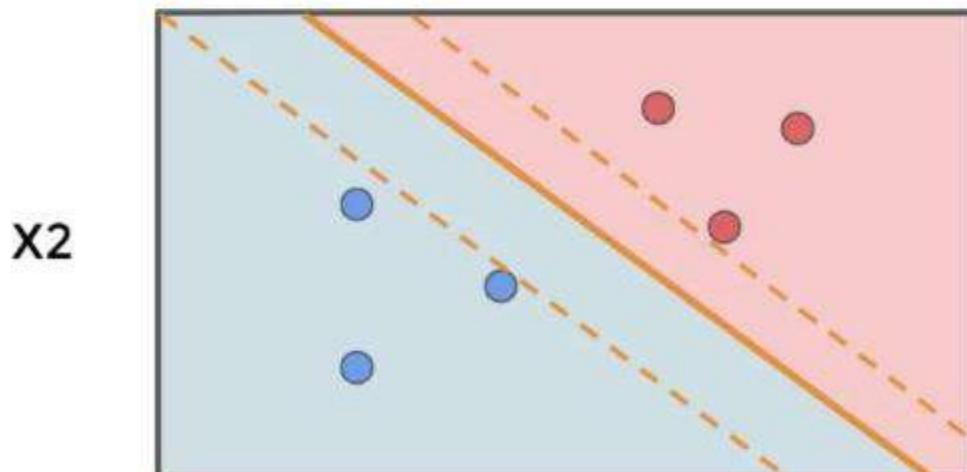
$$a \cdot b = |a||b|\cos(\theta)$$



# Support Vector Machines

- Linear Support Vector Classifier rewritten:

$$f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$$

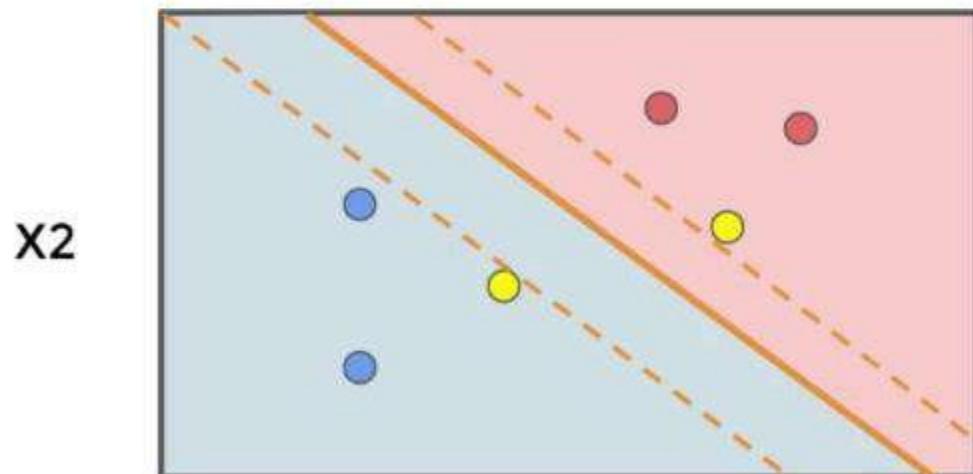


Calculating the inner products of all pairs of training observations

# Support Vector Machines

- Linear Support Vector Classifier rewritten:

$$f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$$

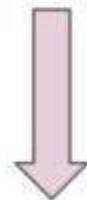


Only non-zero for the support vectors.

# Support Vector Machines

- Linear Support Vector Classifier rewritten:

$$f(x) = \beta_0 + \sum_{i=1}^n \alpha_i \langle x, x_i \rangle$$



$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

# Support Vector Machines

- Linear Support Vector Classifier rewritten:

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

**S** collection of indices of these support points

# Support Vector Machines

- Kernel Function

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij}x_{i'j}$$

A kernel is a function that quantifies the similarity of two observations.

# Support Vector Machines

- Kernel Function

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j} \quad f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

# Support Vector Machines

- Kernel Function

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j}$$

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$

# Support Vector Machines

- Kernel Function

$$K(x_i, x_{i'}) = \sum_{j=1}^p x_{ij} x_{i'j}$$

$$f(x) = \beta_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

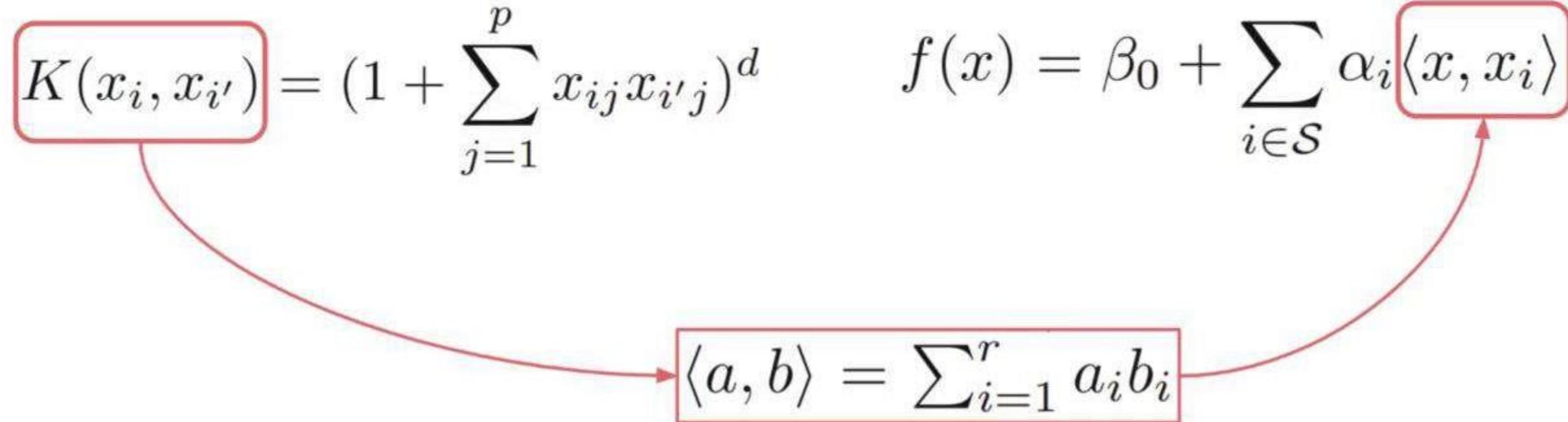
$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$

# Support Vector Machines

- Polynomial Kernel

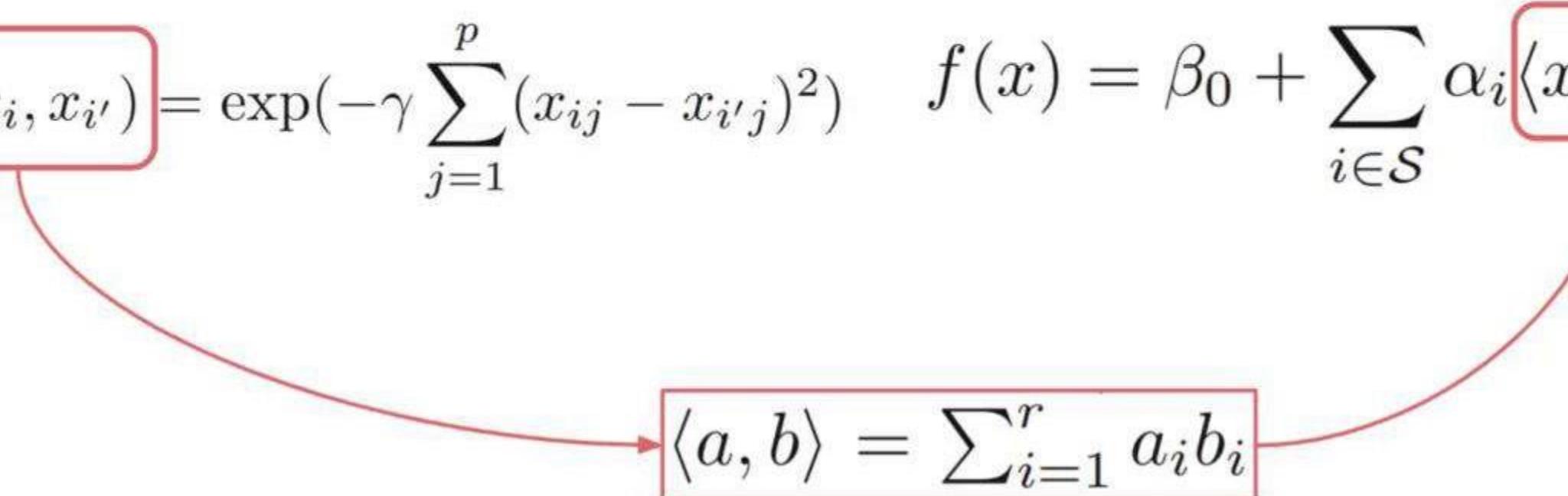
$$K(x_i, x_{i'}) = (1 + \sum_{j=1}^p x_{ij} x_{i'j})^d \quad f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$

$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$



# Support Vector Machines

- Radial Basis Kernel

$$K(x_i, x_{i'}) = \exp(-\gamma \sum_{j=1}^p (x_{ij} - x_{i'j})^2)$$
$$f(x) = \beta_0 + \sum_{i \in \mathcal{S}} \alpha_i \langle x, x_i \rangle$$
$$\langle a, b \rangle = \sum_{i=1}^r a_i b_i$$


# Support Vector Machines

- The use of **kernels** as a replacement is known as the **kernel trick**.
- Kernels allow us to avoid computations in the enlarged feature space, by only needing to perform computations for each distinct pair of training points (details in 9.3.2 in ISLR).

# Support Vector Machines

- Intuitively we've already seen inner products act as a measurement of similarity between vectors.
- The use of kernels can be thought of as a measure of similarity between the original feature space and the enlarged feature space.

# Support Vector Machines

- Now that we understand the theory and intuition behind SVMs, let's move on to actually using them with code!

# Support Vector Machines

Classification with Scikit-Learn - Part One

# Support Vector Machines

- Note we will use a function from a .py file included in the Support Vector Machine folder!



00-SVM-Classification.ipynb

# Support Vector Machines

Classification with Scikit-Learn - Part Two



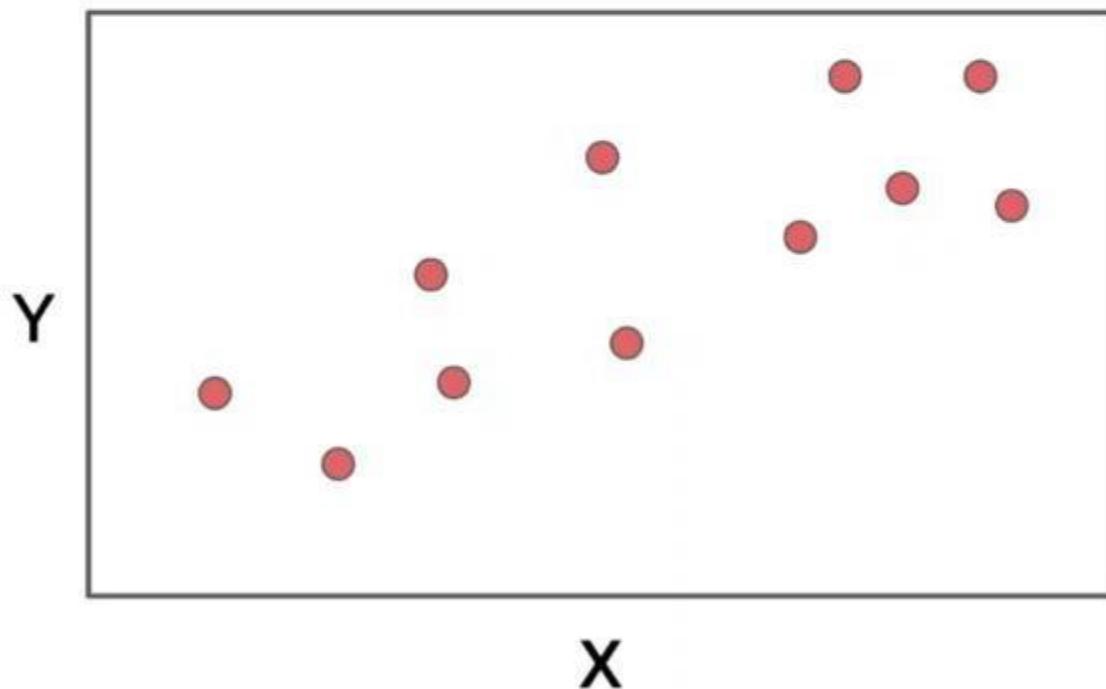
00-SVM-Classification.ipynb

# Support Vector Machines

Regression with Scikit-Learn - Part One

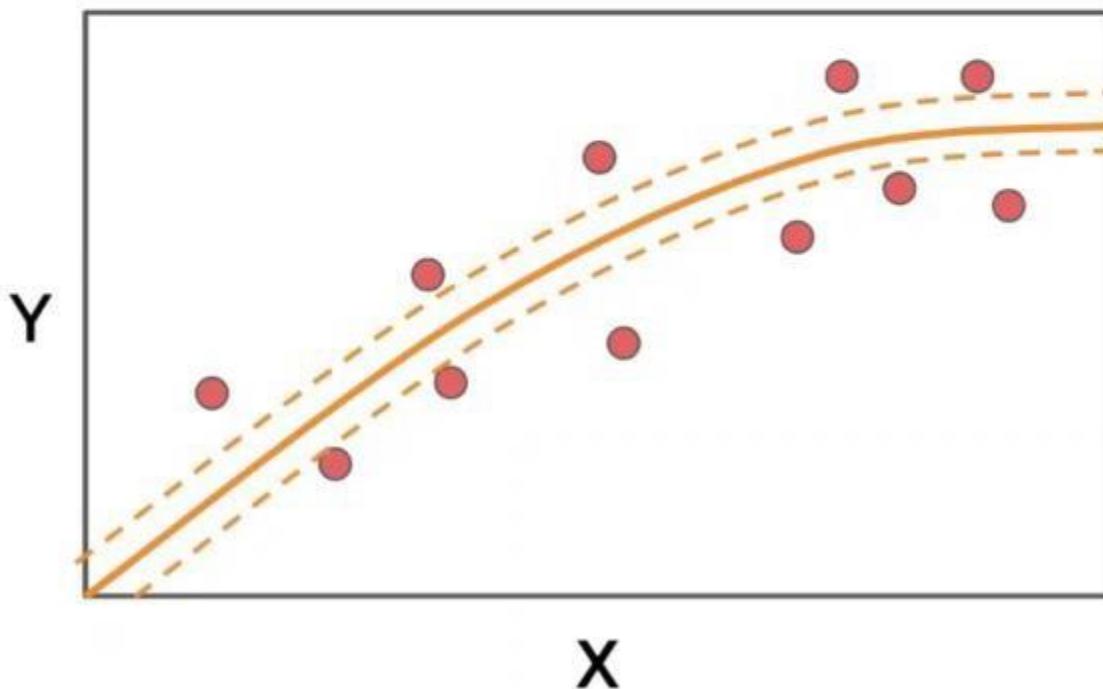
# Support Vector Machines

- Support Vector Regression



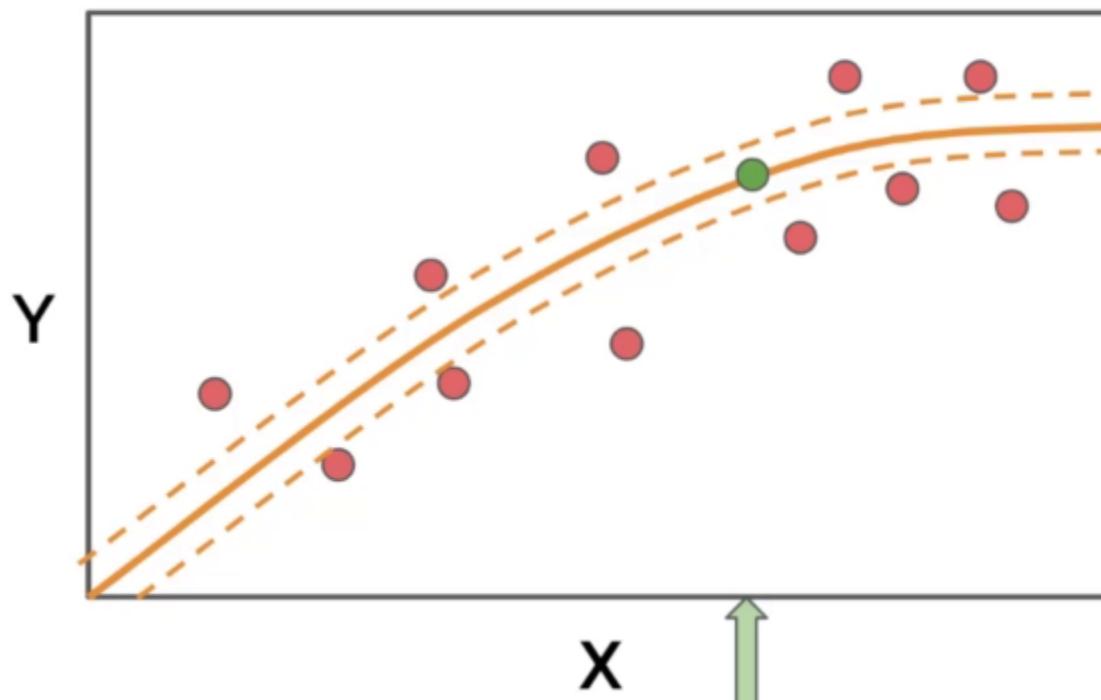
# Support Vector Machines

- Support Vector Regression



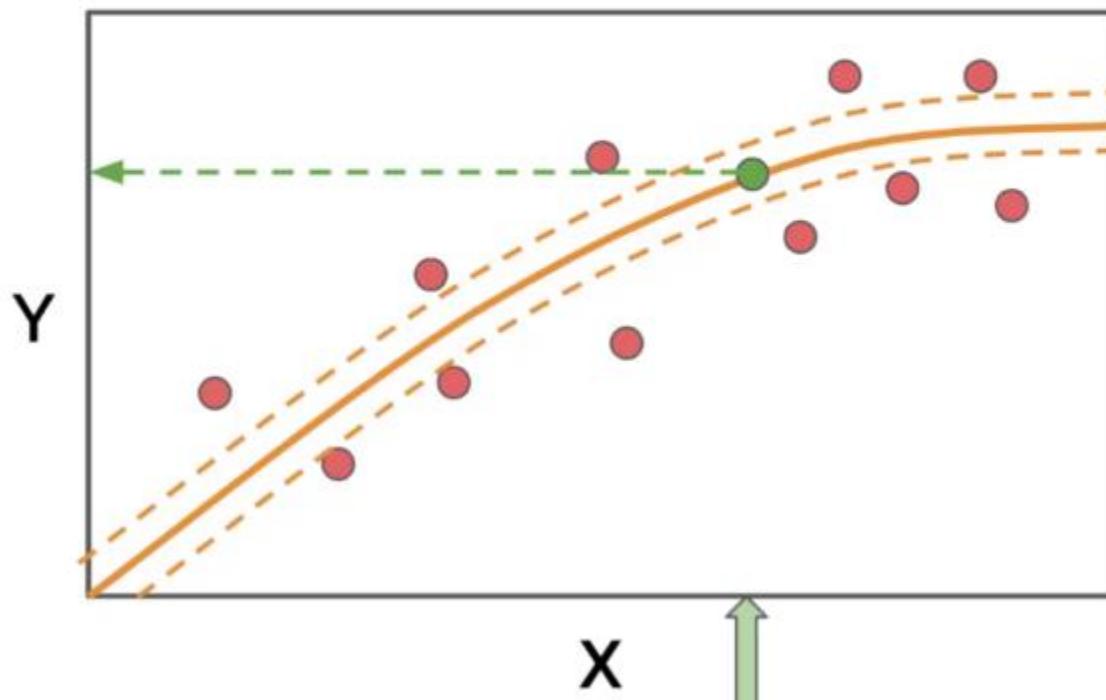
# Support Vector Machines

- Support Vector Regression



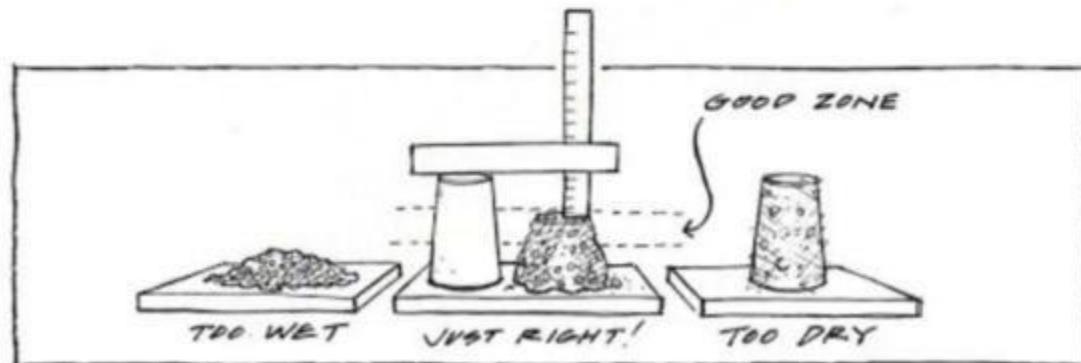
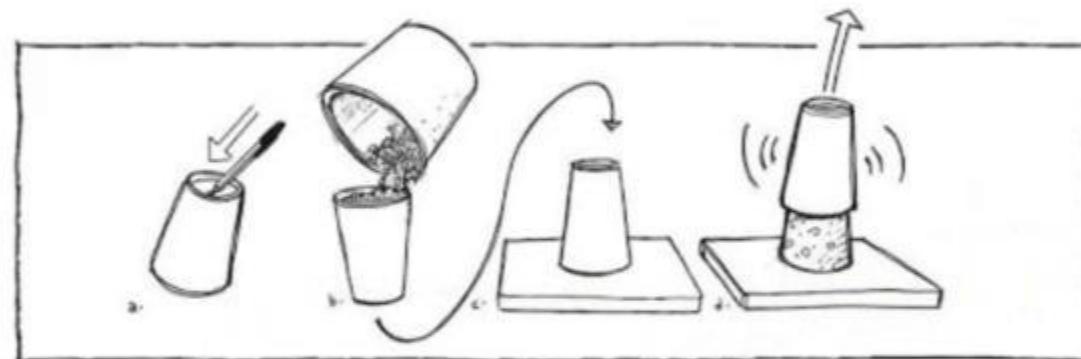
# Support Vector Machines

- Support Vector Regression



# Support Vector Machines

- Concrete Slump Test





# Support Vector Machines

- Concrete Slump Test



# Support Vector Machines

- Concrete Slump Test





01-SVM-Regression.ipynb

# Support Vector Machines

# Project Exercise

# Support Vector Machines

- Can you detect fraudulent wine based on features from a chemical analysis?



# Support Vector Machines

- Is this problem even solvable? Are “fraud” wines really different than “real” wines?





02-SVM-Project-Exercise.ipynb