An Introduction to Machine Learning with Python Programming

11 Sep 2023 - 20 Oct 2023

Conducted by:

iHUB Divya Sampark, IIT Roorkee

and

Support Vector Machines

Presented by:

Shreyas Shukla

Does a hyperplane exist that can effectively separate classes?

Support Vector Machines

Theory and Intuition - Hyperplanes and Margins

We will slowly reach up to SVMs:

- Maximum Margin Classifier
- Support Vector Classifier
- Support Vector Machines

Let’s begin by understanding what is a **hyperplane**.

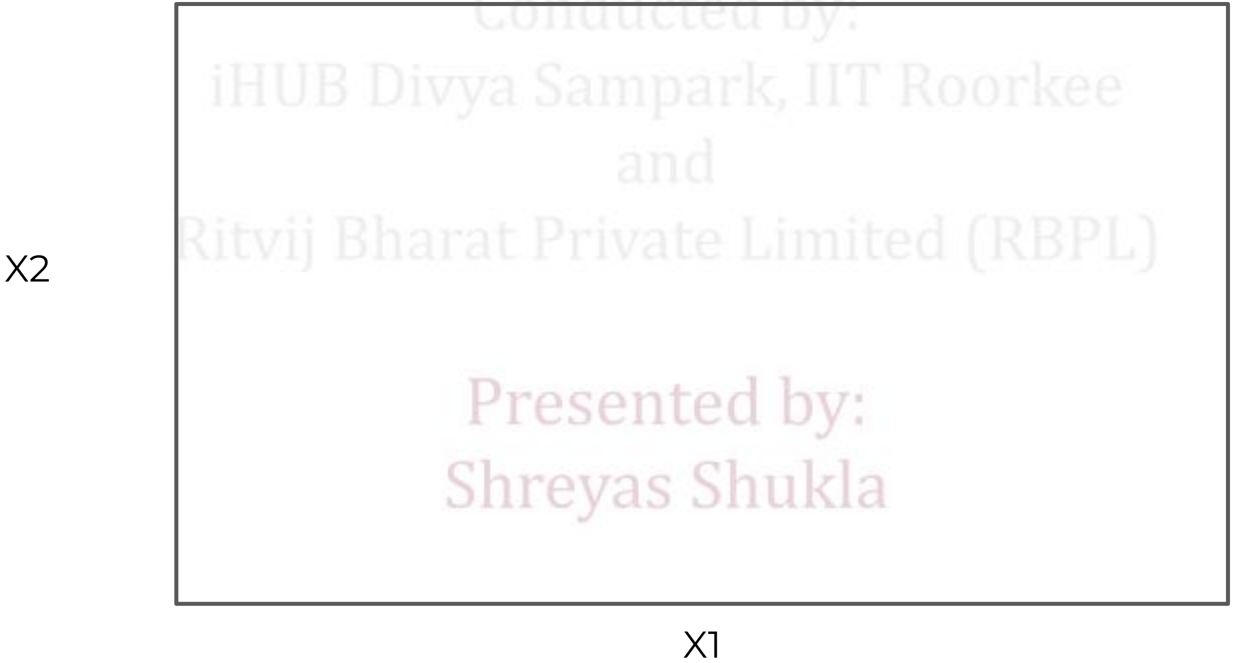
In a space with N dimensions, a hyperplane can be understood as a flat subspace that has a dimension of N - 1, and it is formed by affine points.

- 1-D Hyperplane is a single point
- 2-D Hyperplane is a line
- 3-D Hyperplane is flat plane

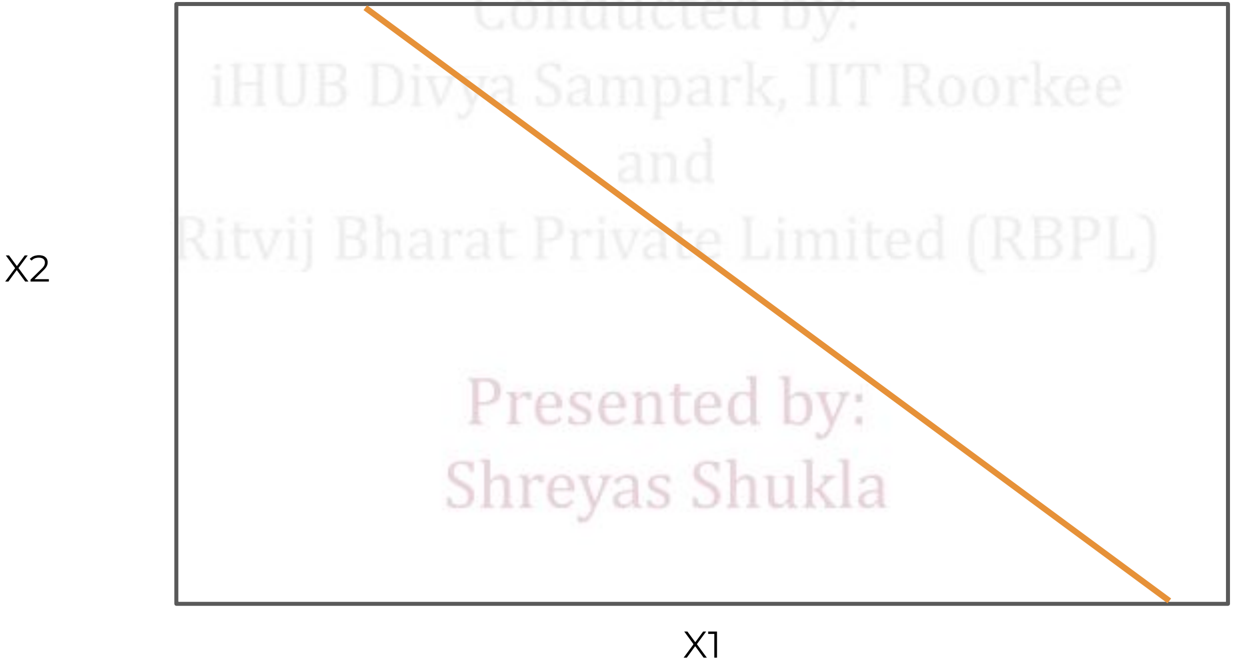
1-D Hyperplane

1-D Hyperplane

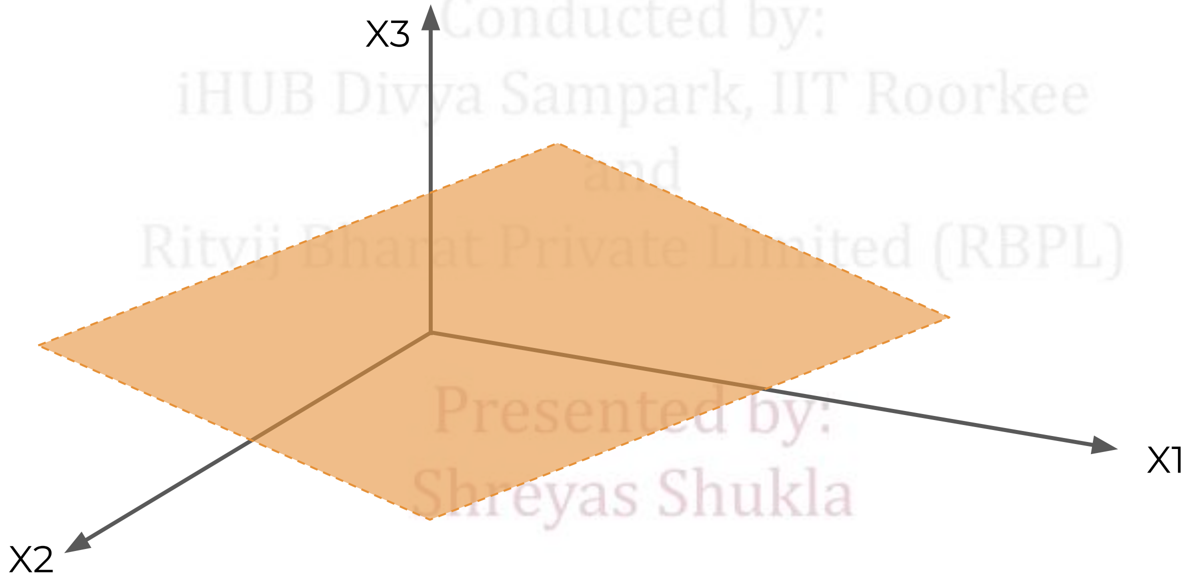
2-D Hyperplane



2-D Hyperplane



3-D Hyperplane



We can use Hyperplanes to create a separation between classes.

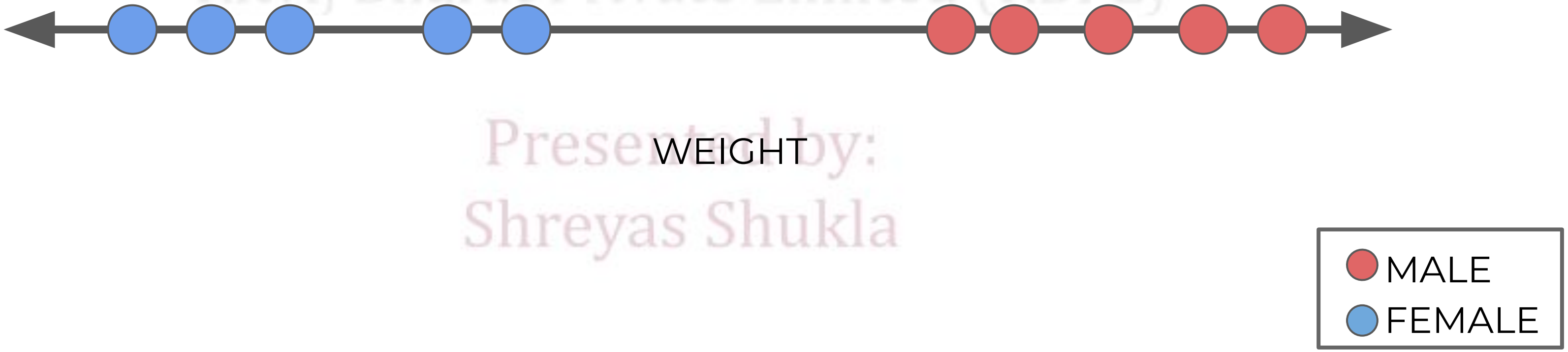
After establishing the separating hyperplane, any new points introduced will be categorized by which side of the hyperplane they fall on, allowing us to assign them to a specific class.

Let us assume a data set with one feature and one binary target label.For example:

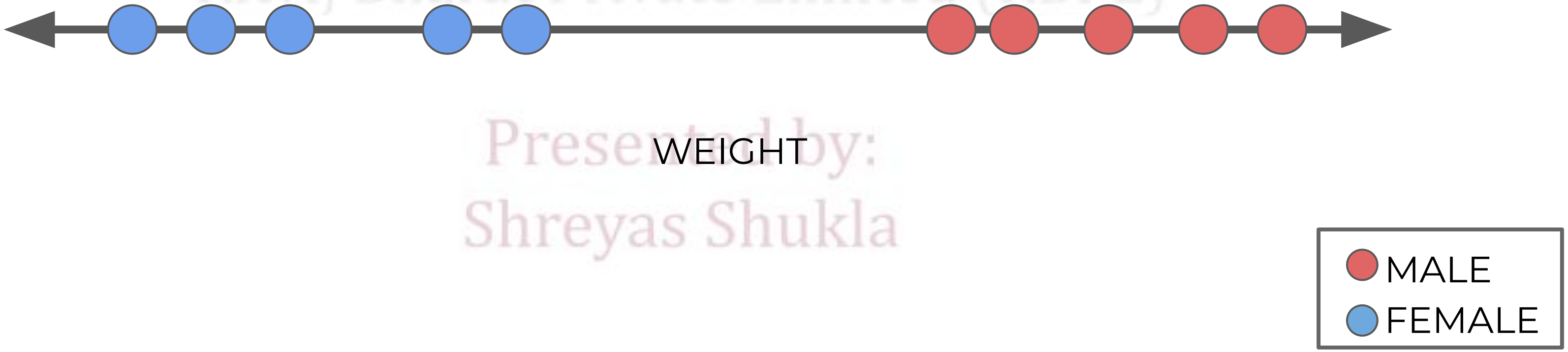
- A weight feature for baby chicks
- Classified by Male or Female

What would this be visualized?

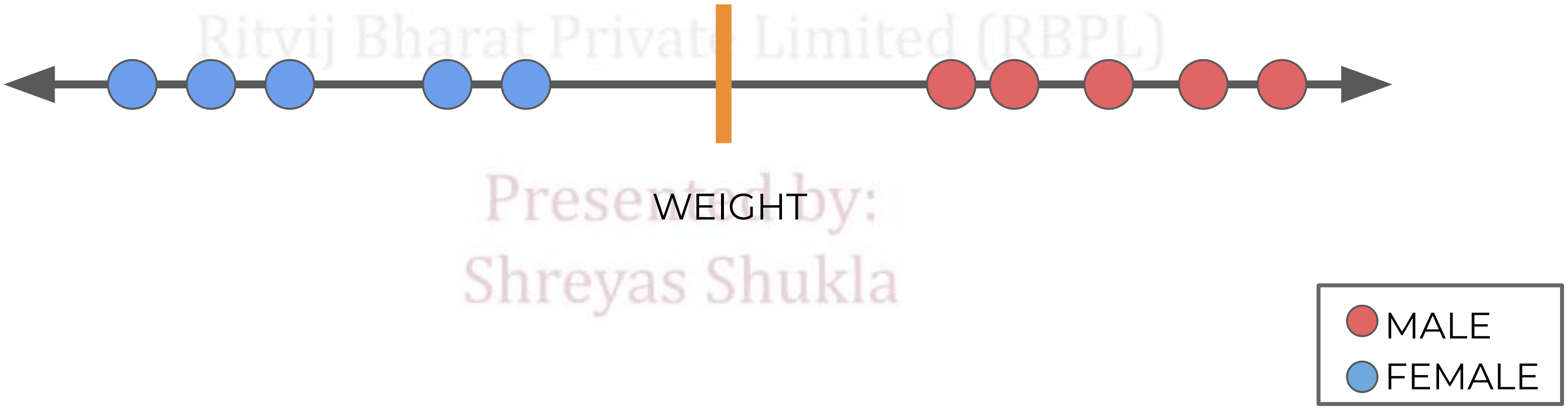
Place points along feature.



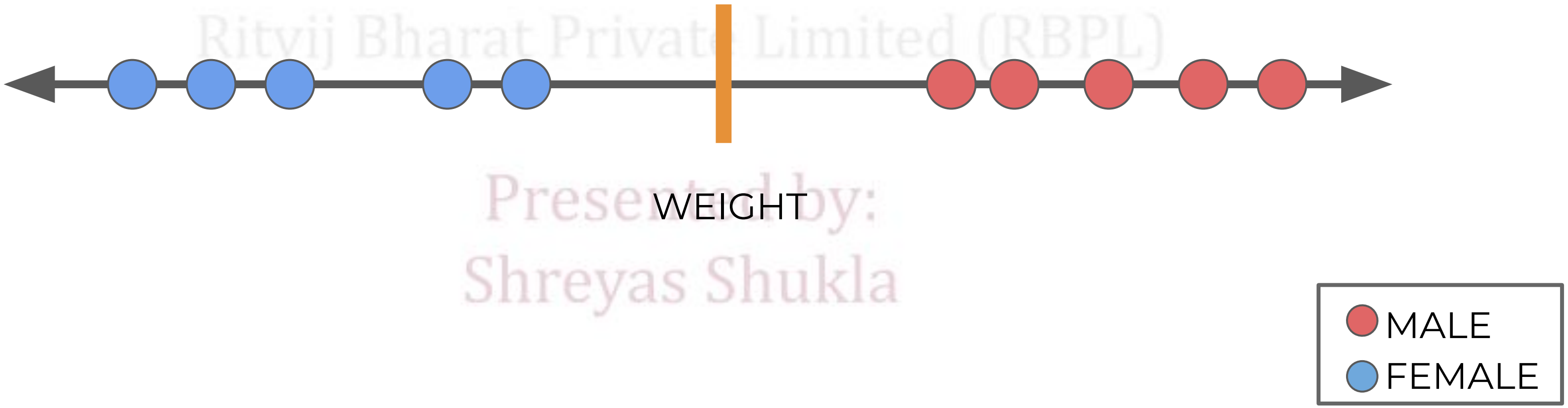
Notice in this case, classes are perfectly separable. This is unlikely in real world datasets.



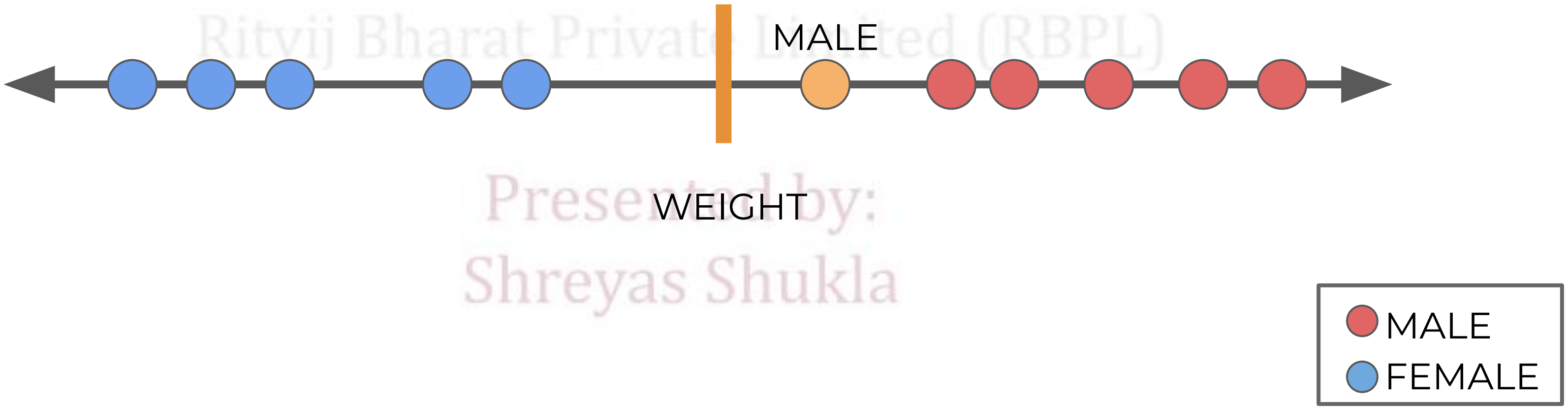
Idea behind SVM is to create a **hyperplane** that will separate the classes.



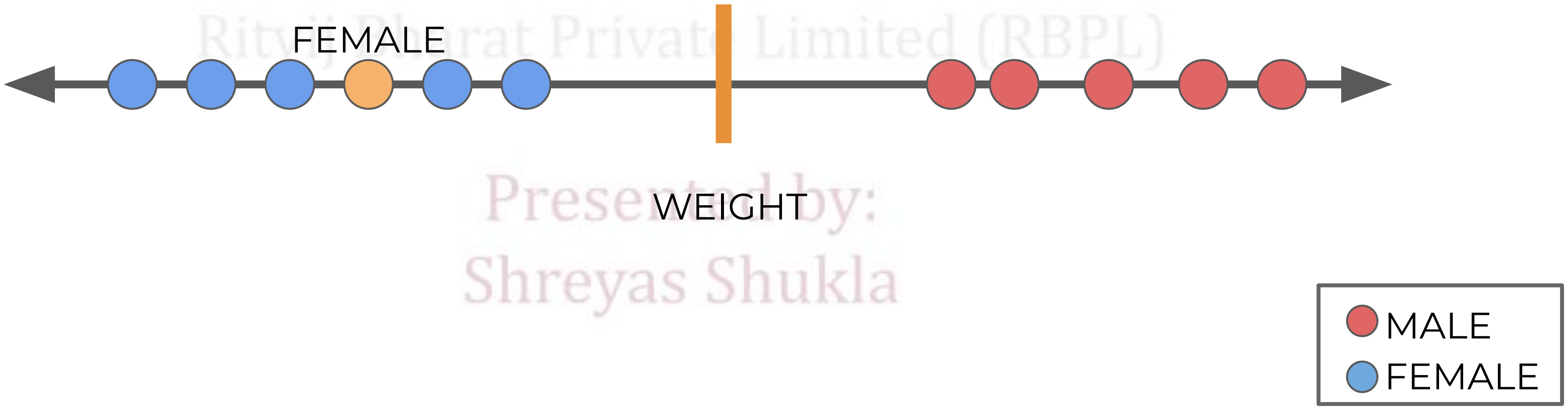
The classification of a new point is determined by the side of the hyperplane on which it falls.



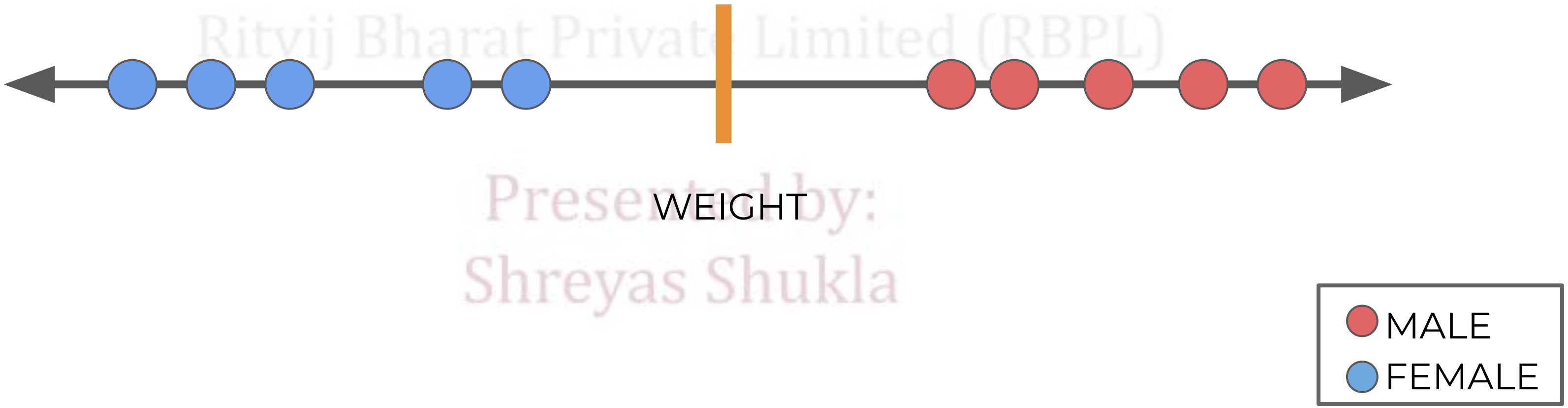
The classification of a new point is determined by the side of the hyperplane on which it falls.



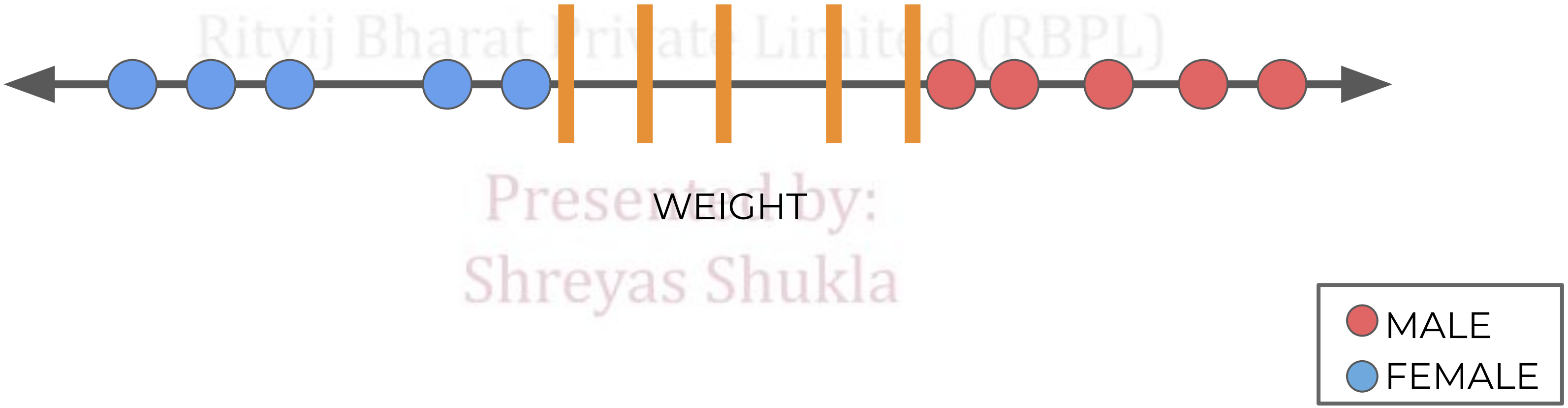
The classification of a new point is determined by the side of the hyperplane on which it falls.



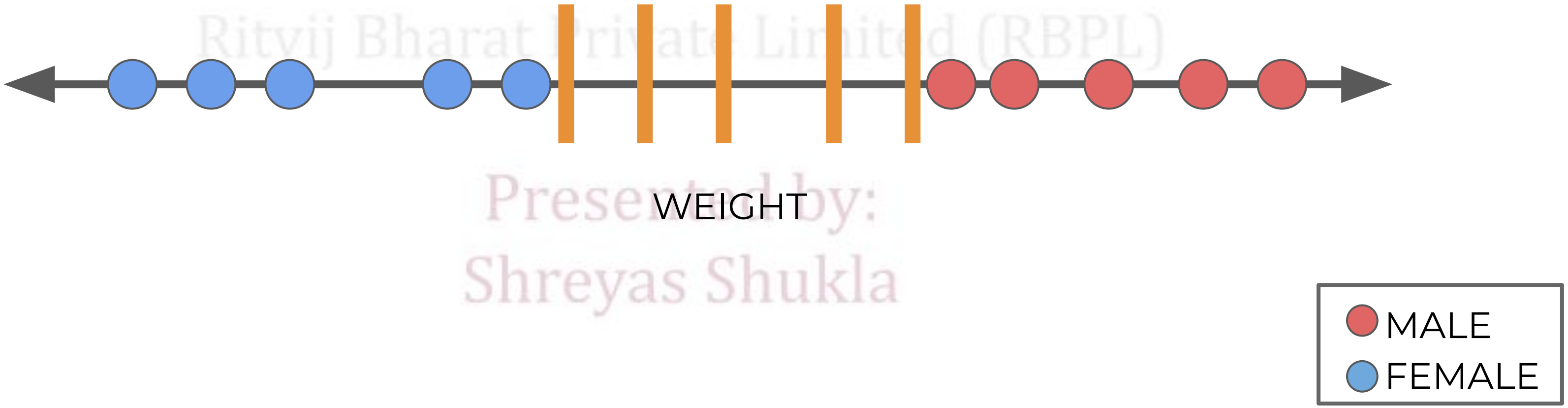
The classification of a new point is determined by the side of the hyperplane on which it falls.



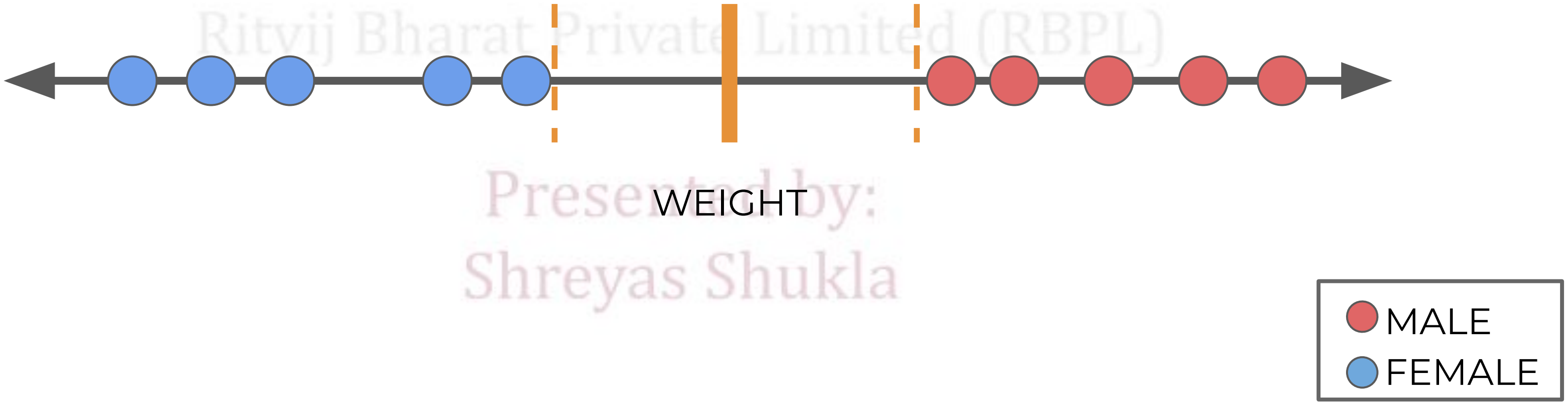
You’ll notice that there are many options that perfectly separate out these classes



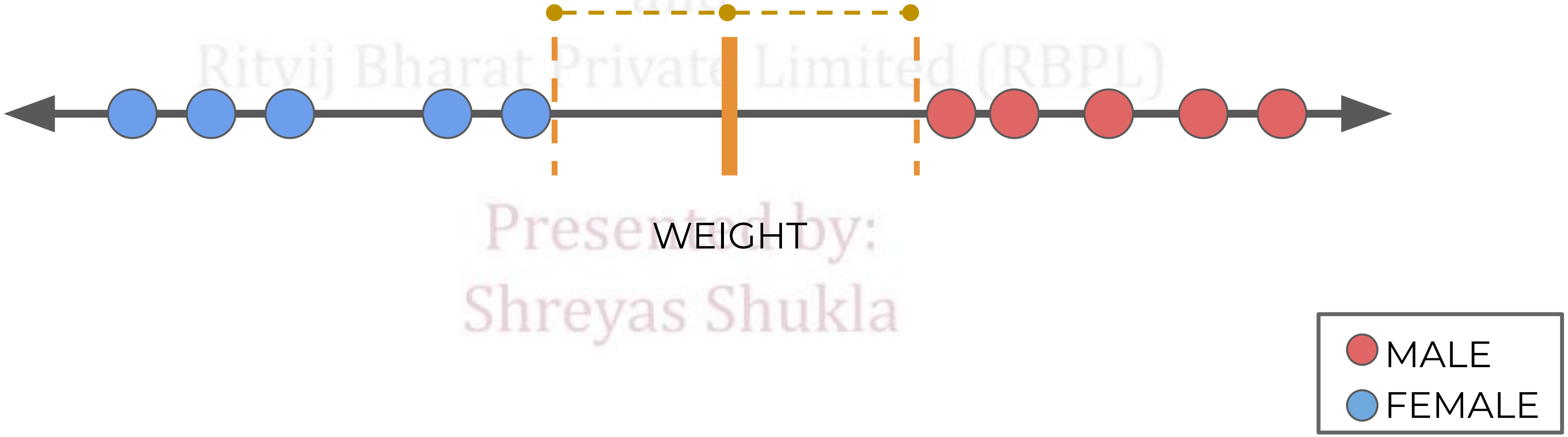
Which one is the “best” separator between the classes?



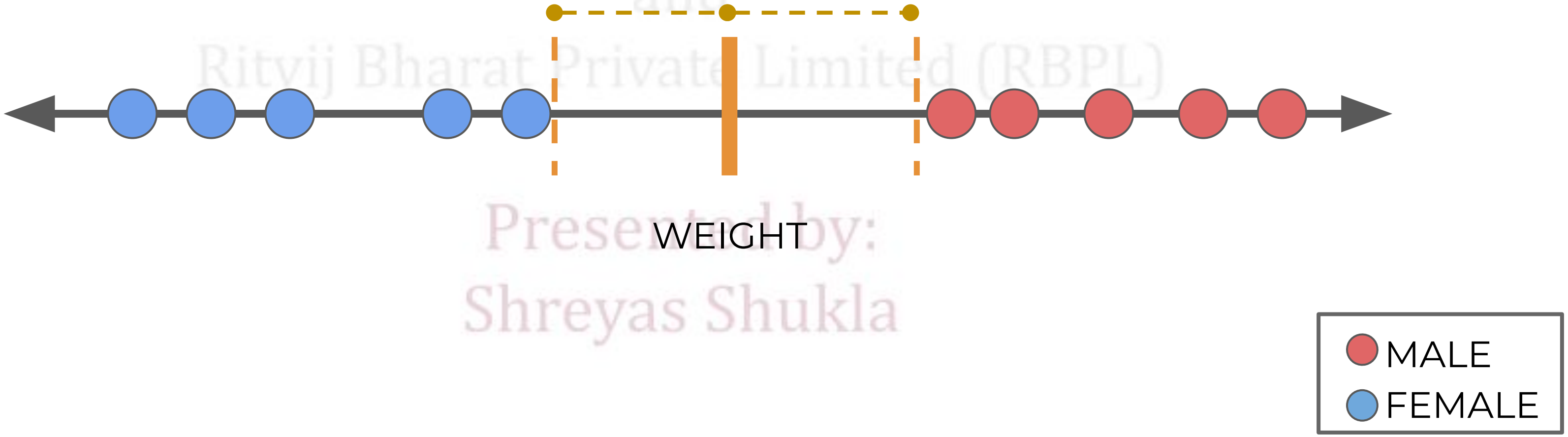
Use the separator that **maximizes** the **margins** between the classes.



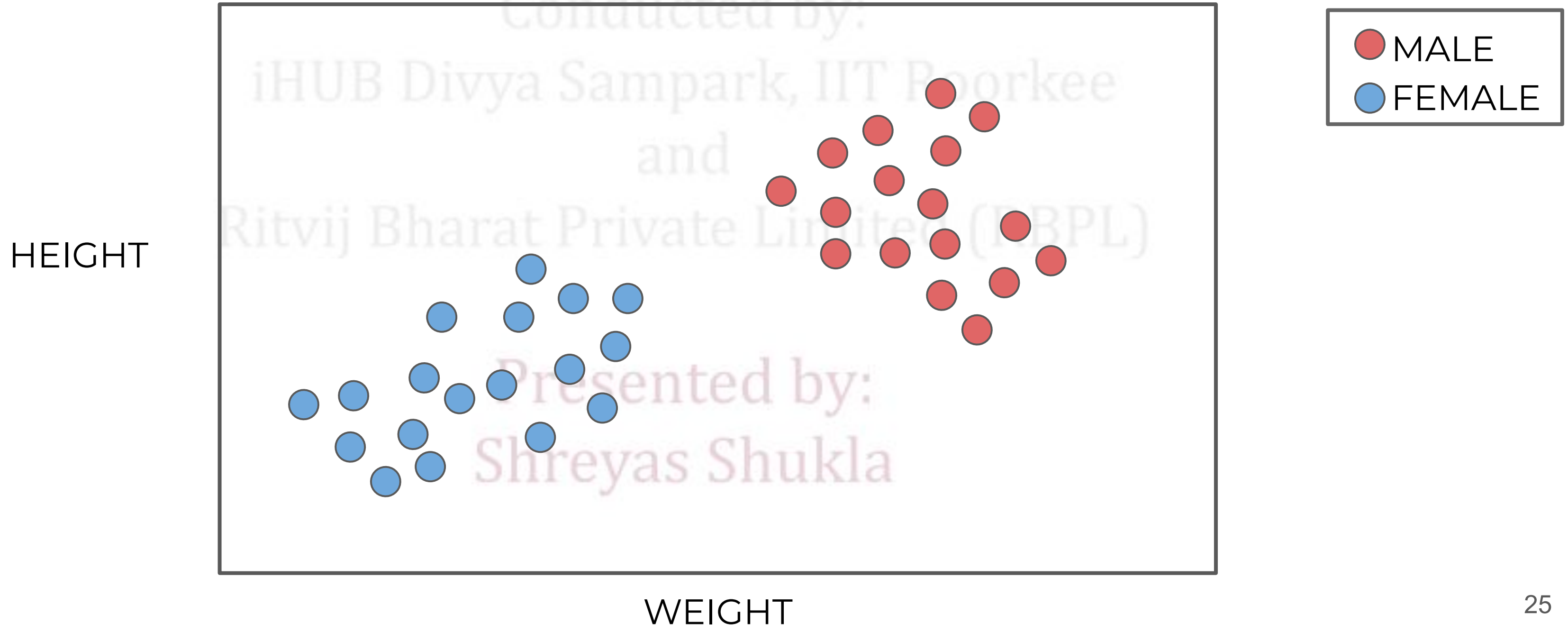
Maximal Margin Classifier.



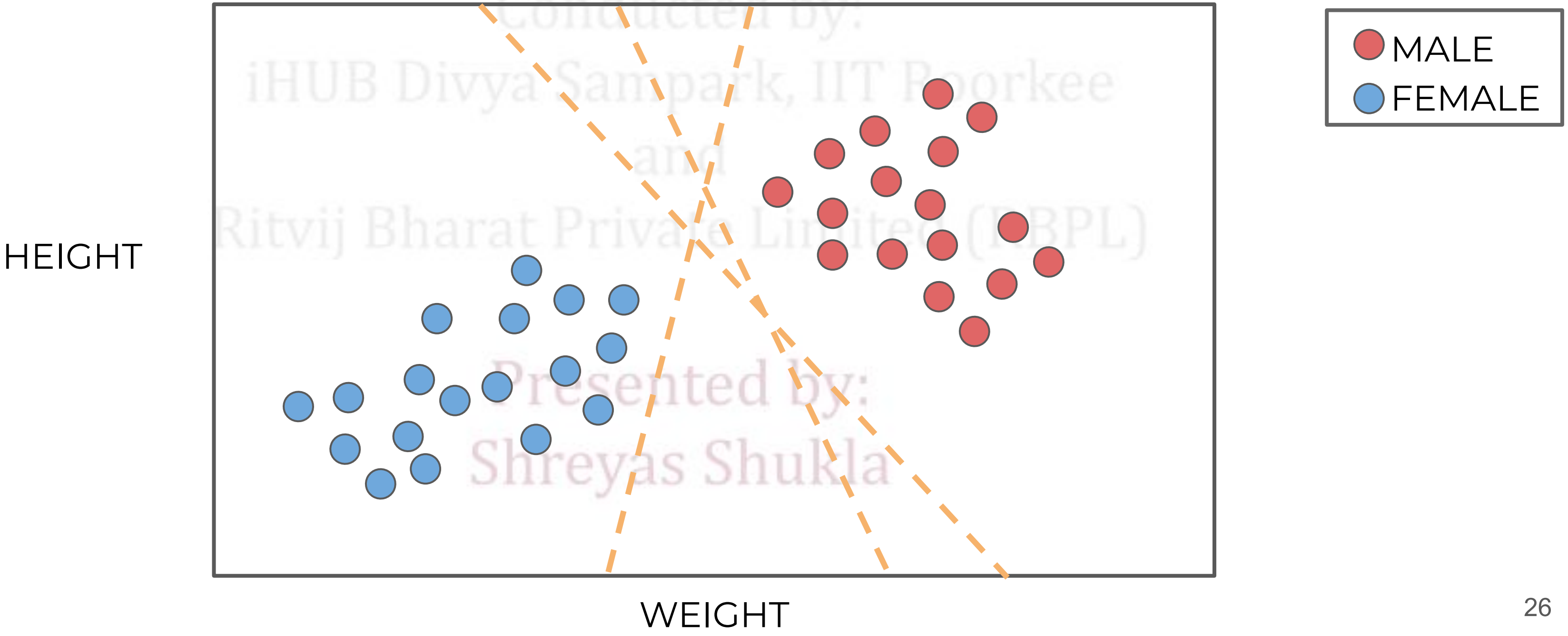
This very idea of maximum margins applies to N-dimensions.



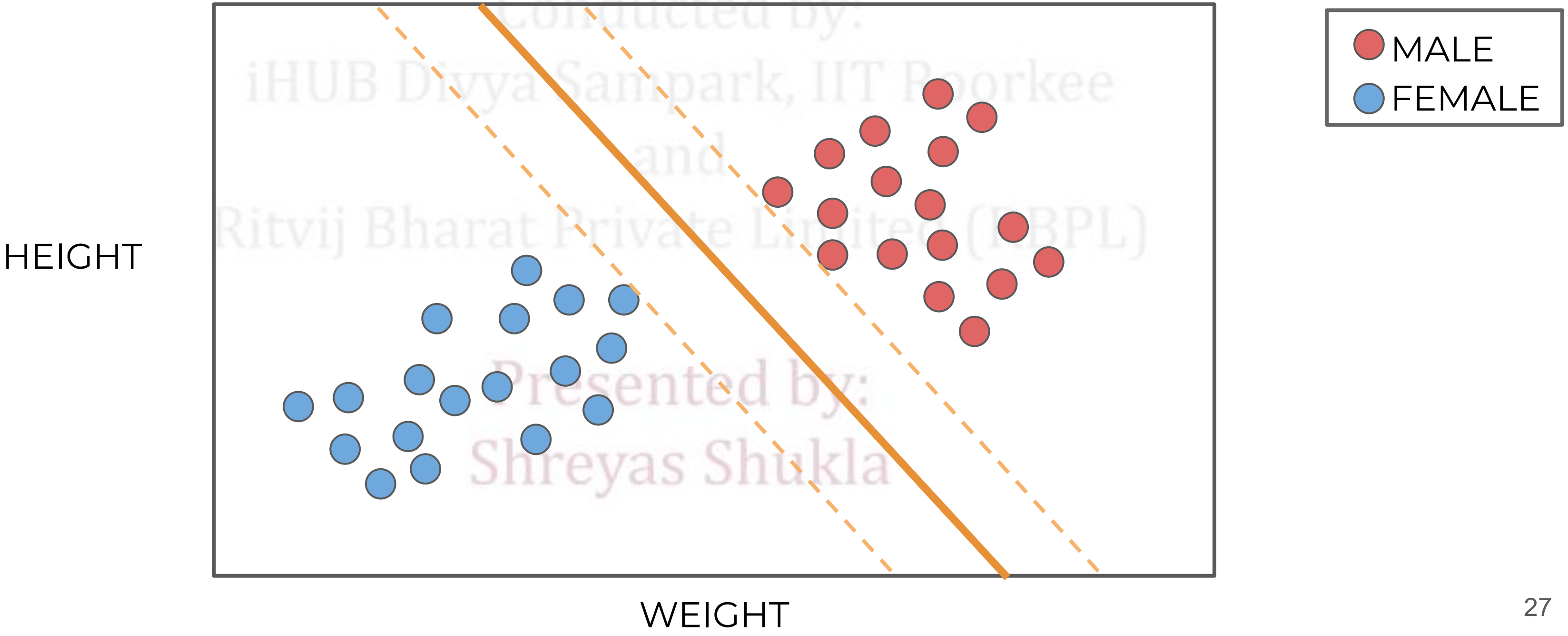
Imagine a 2 dimensional feature space:



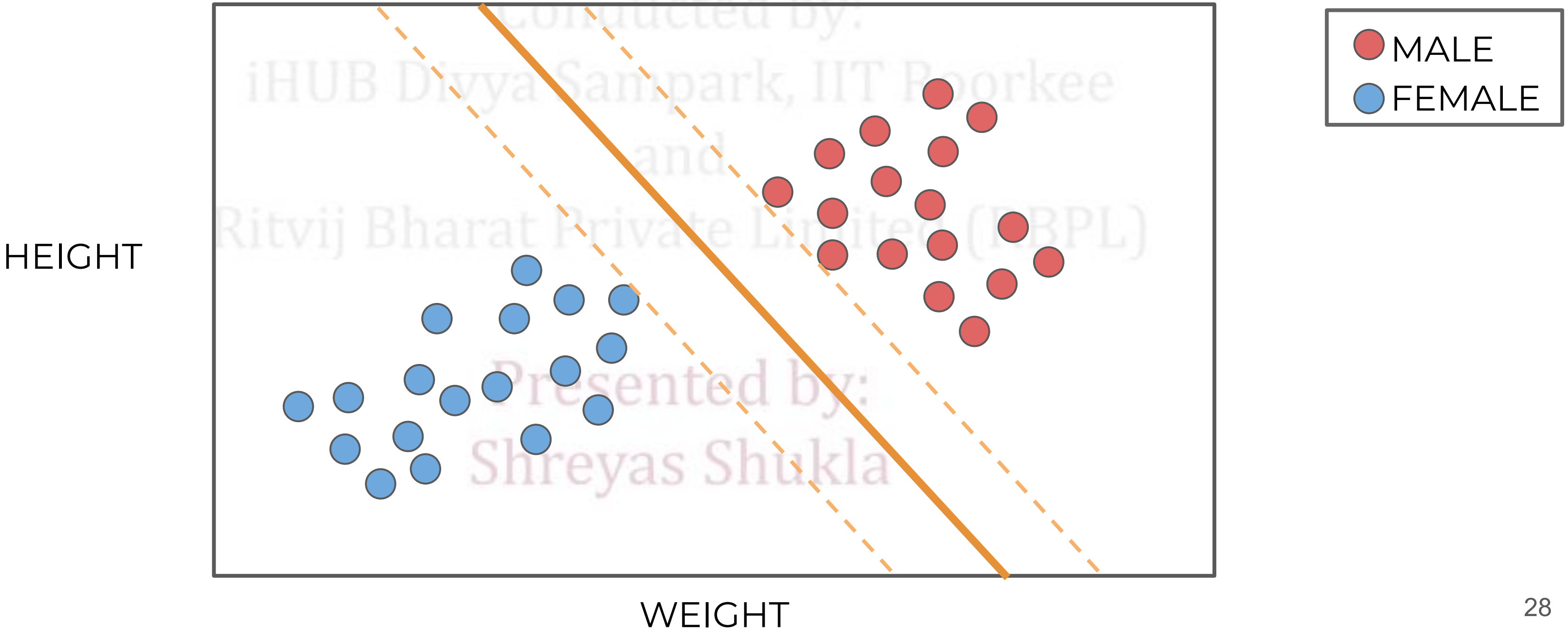
We could have Multiple possible hyperplanes:



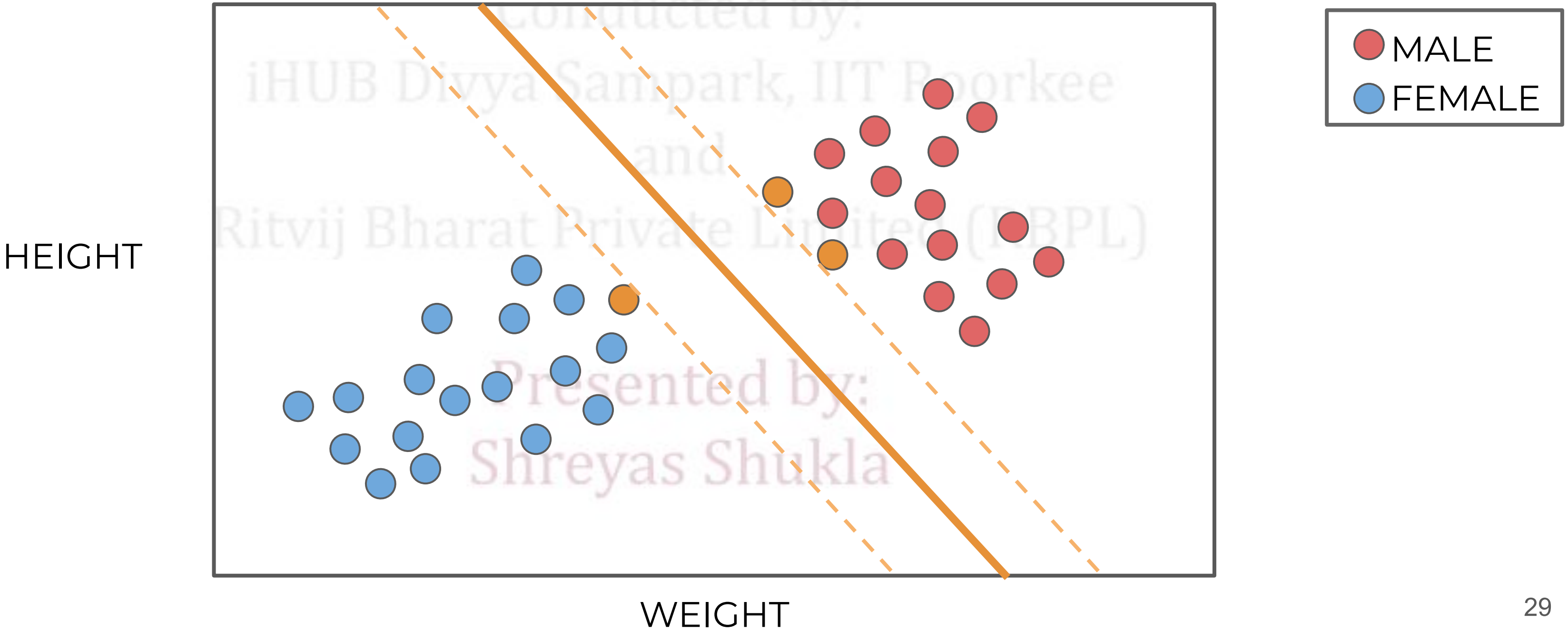
Choose to maximize margins:



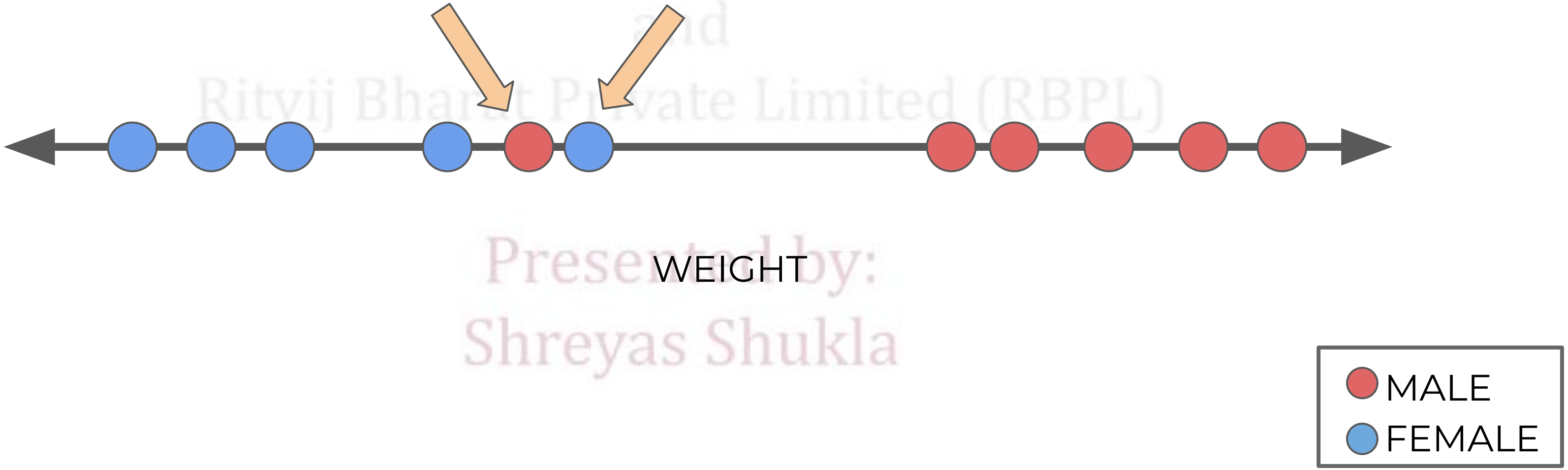
Note each data point is a 2D vector:



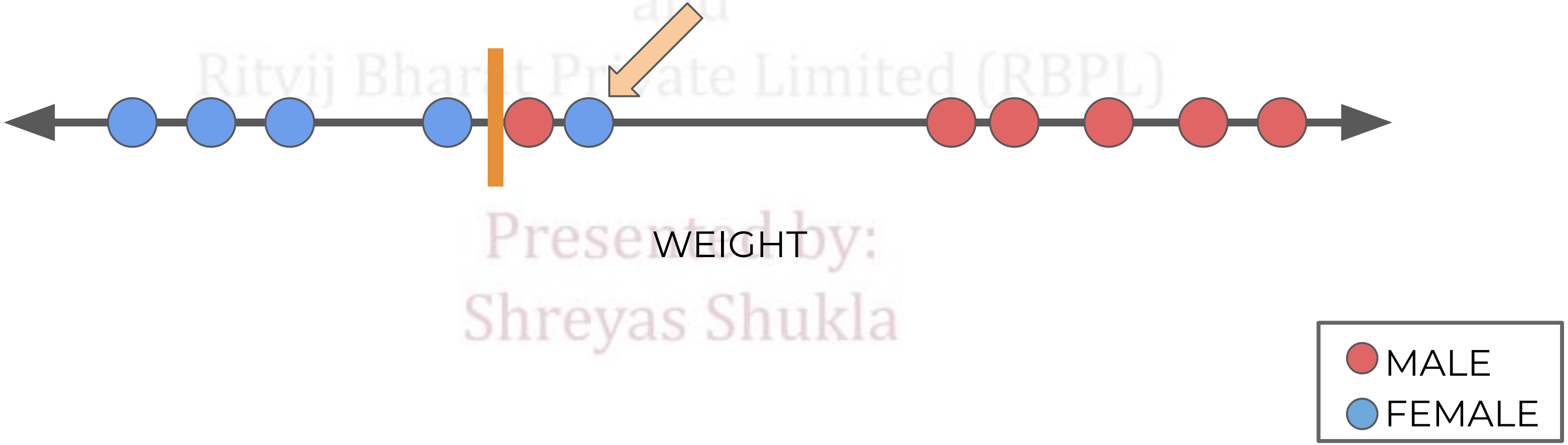
Data points at margin support separator:



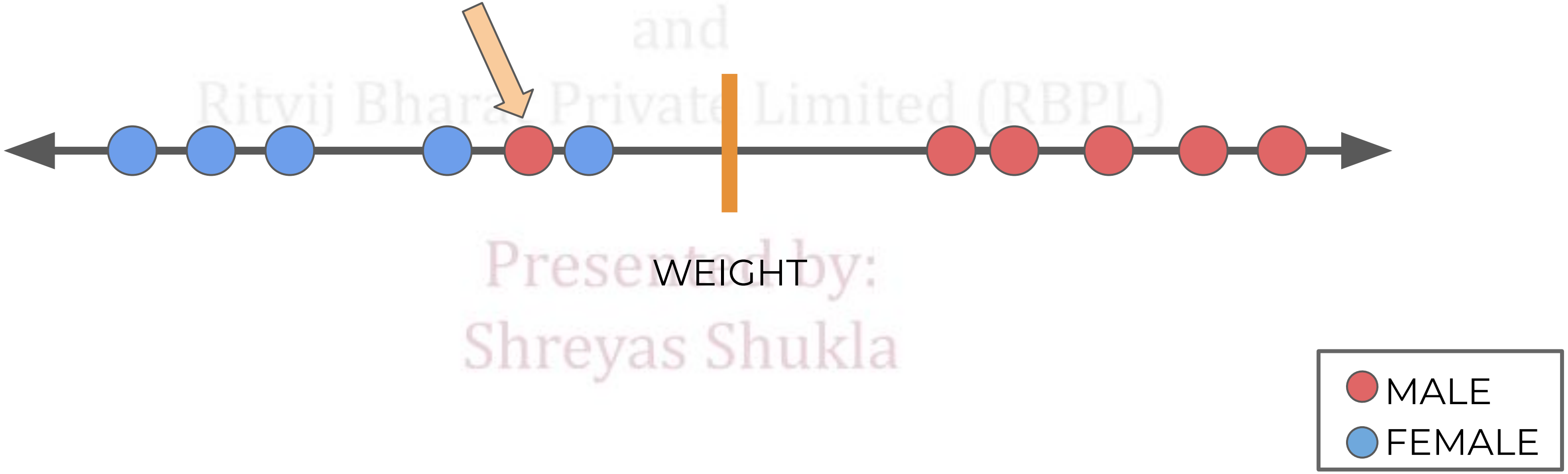
What happens if classes are not perfectly separable?



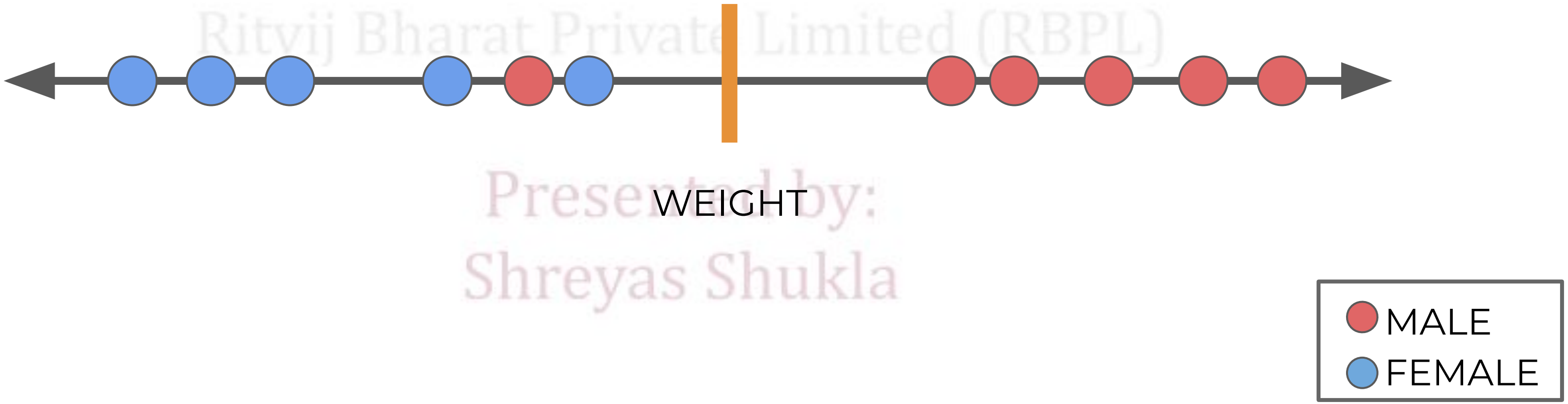
It is impossible to achieve perfect separation without accepting the possibility of misclassifications.



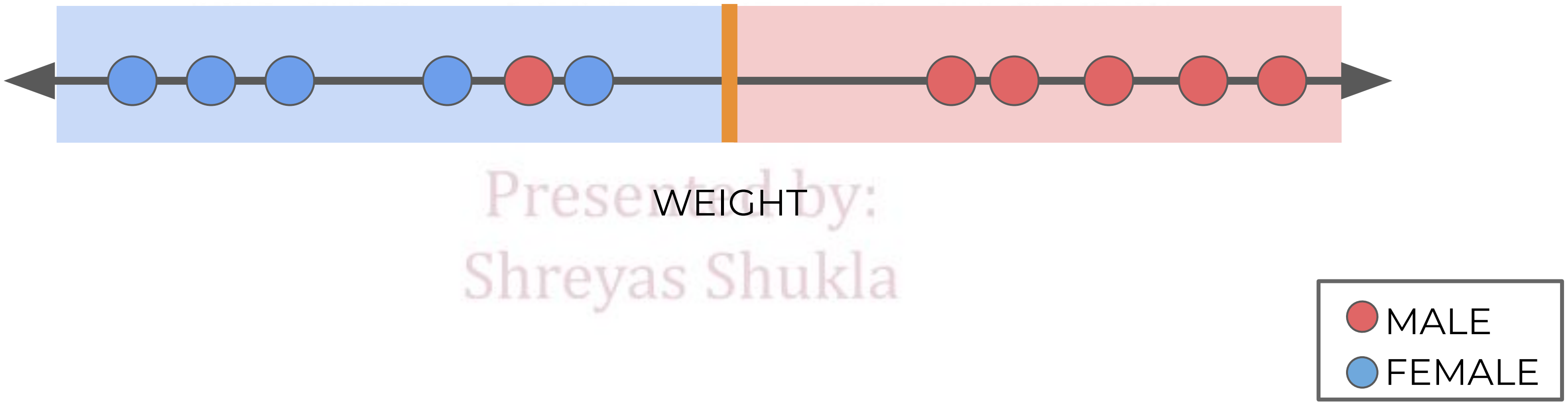
It is impossible to achieve perfect separation without accepting the possibility of misclassifications.



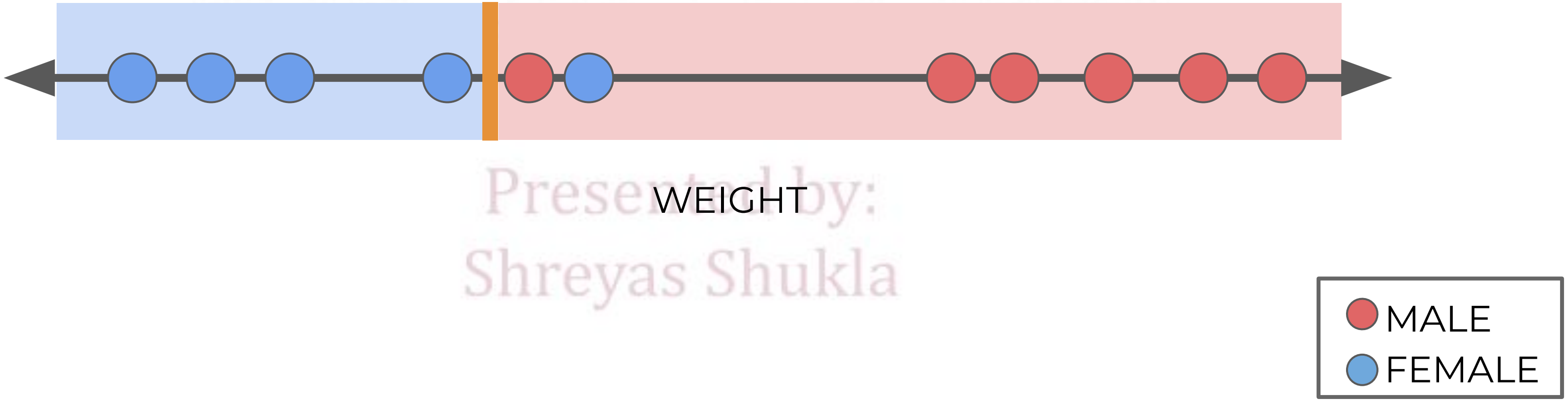
We encounter a bias-variance trade-off depending where we place this separator:



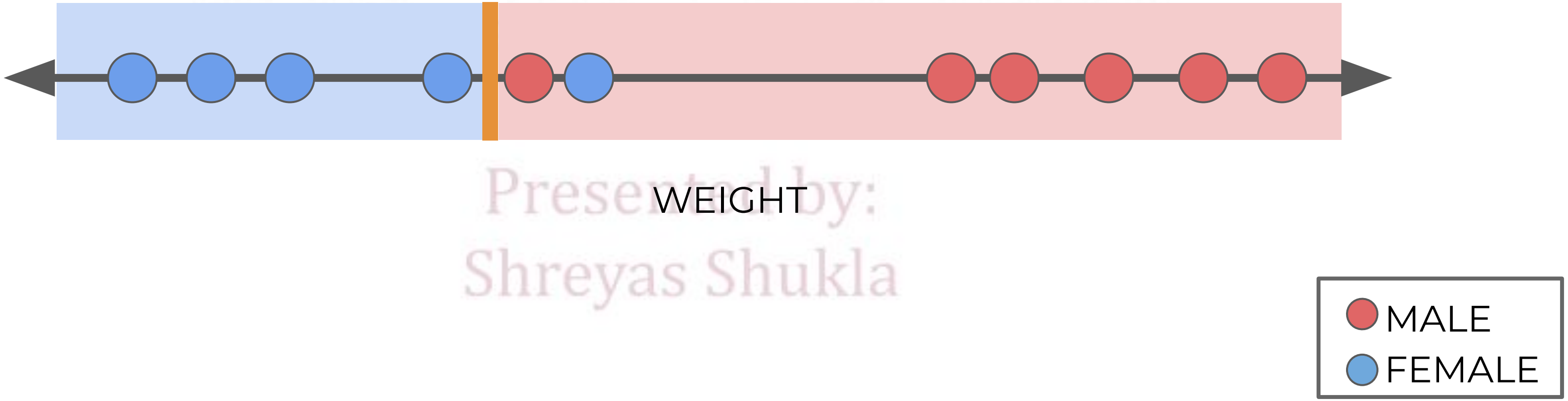
For one feature this classifier creates range for male and female:



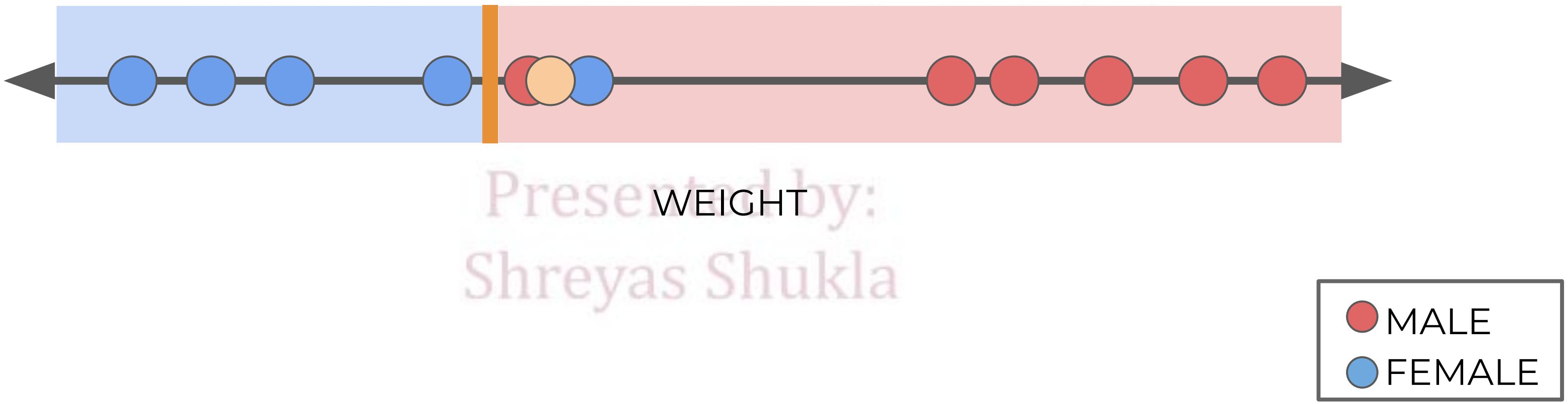
This fit only misclassified one female training point as male:

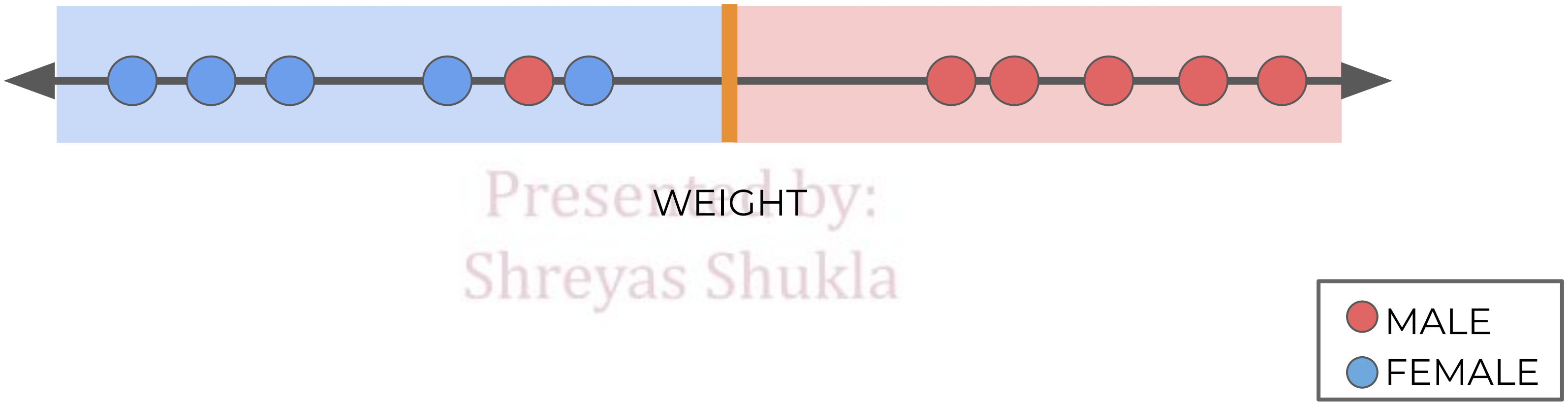


This is a high variance fit to training data, picking too much noise from Female:

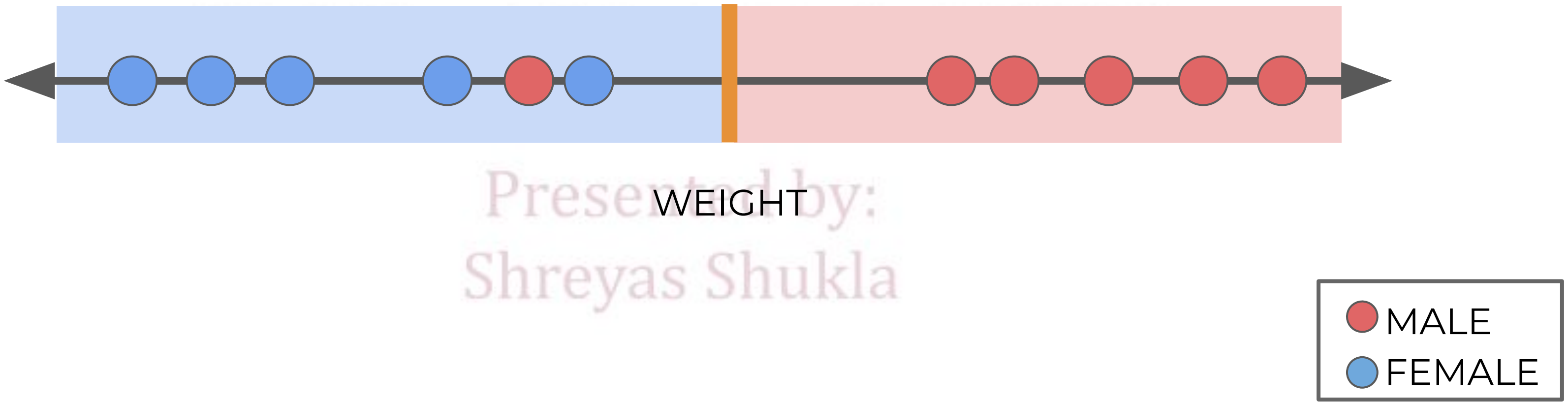


A new test point close to existing female weights could get classified as male:

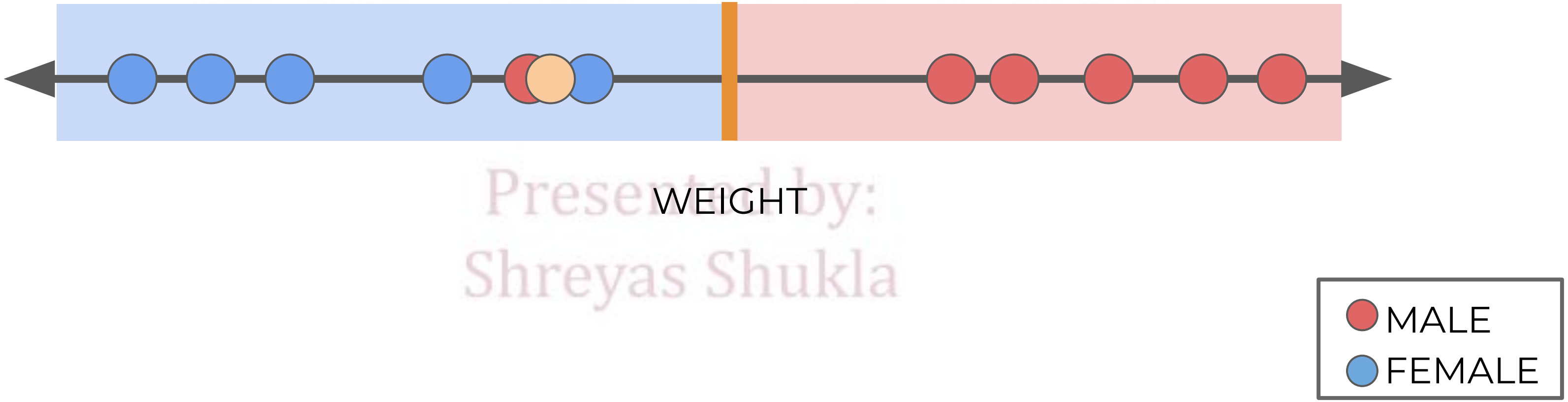



WEIGHT
MALE
FEMALE

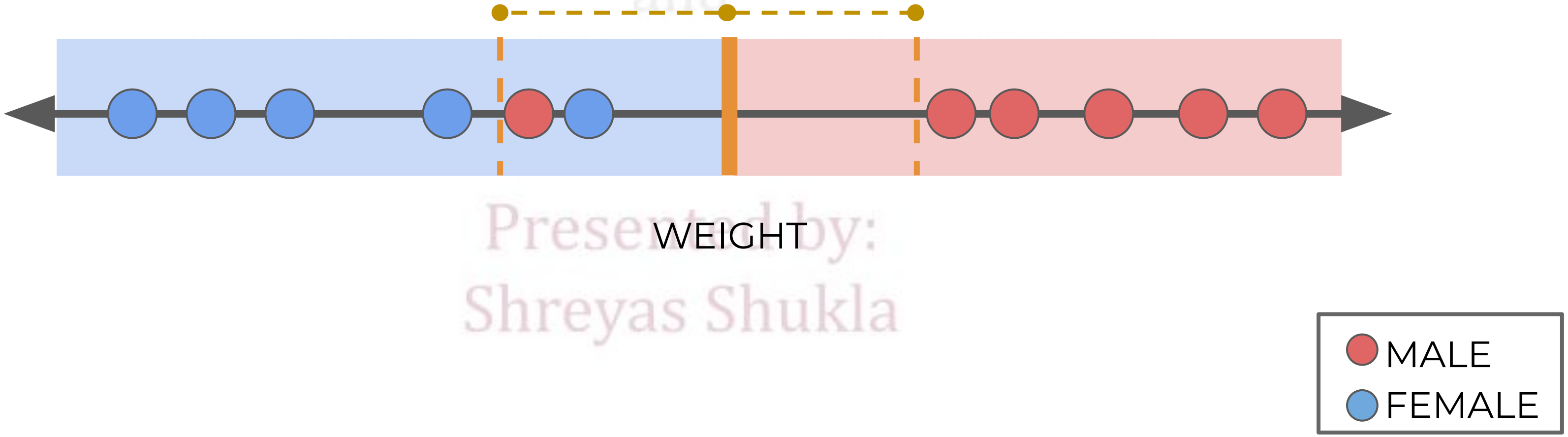
We allow more bias to achieve better long term results on future data:



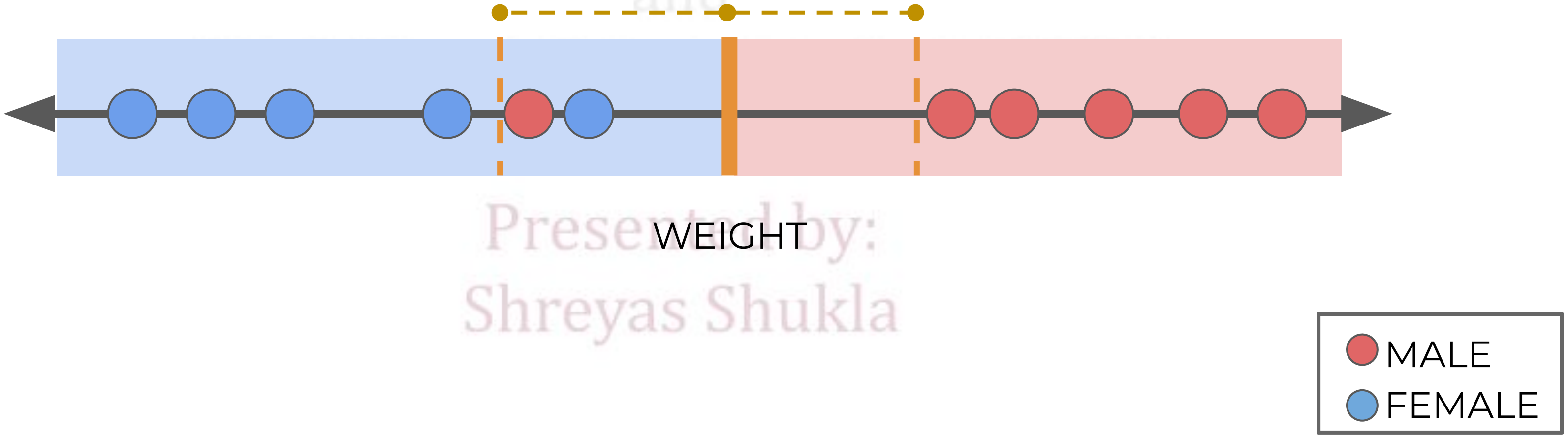
We allow more bias to achieve better long term results on future data:



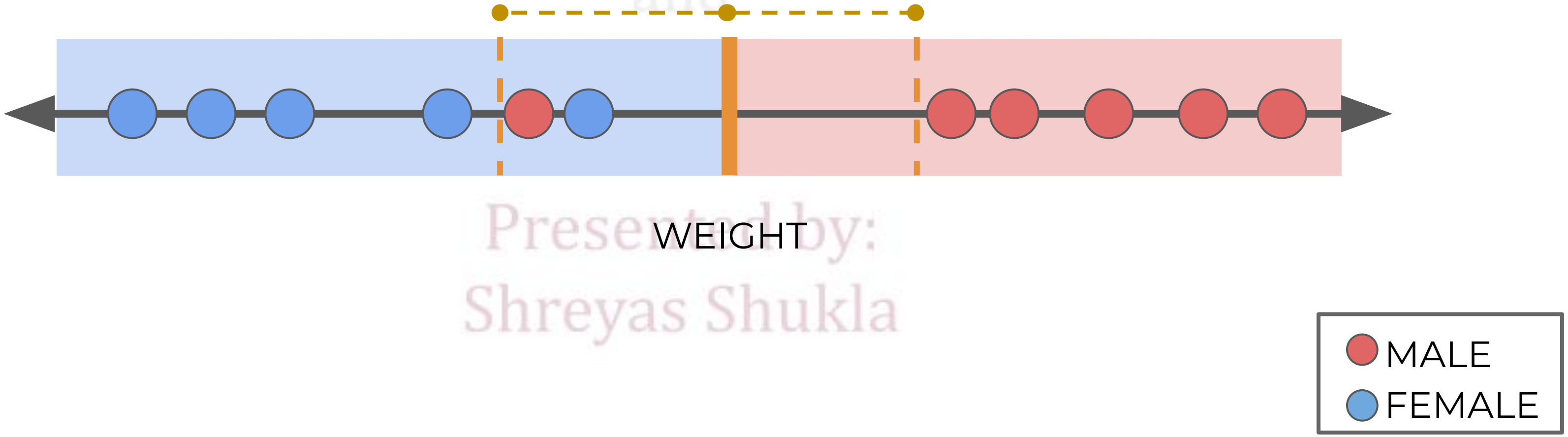
Soft margin: Distance between threshold and the observations



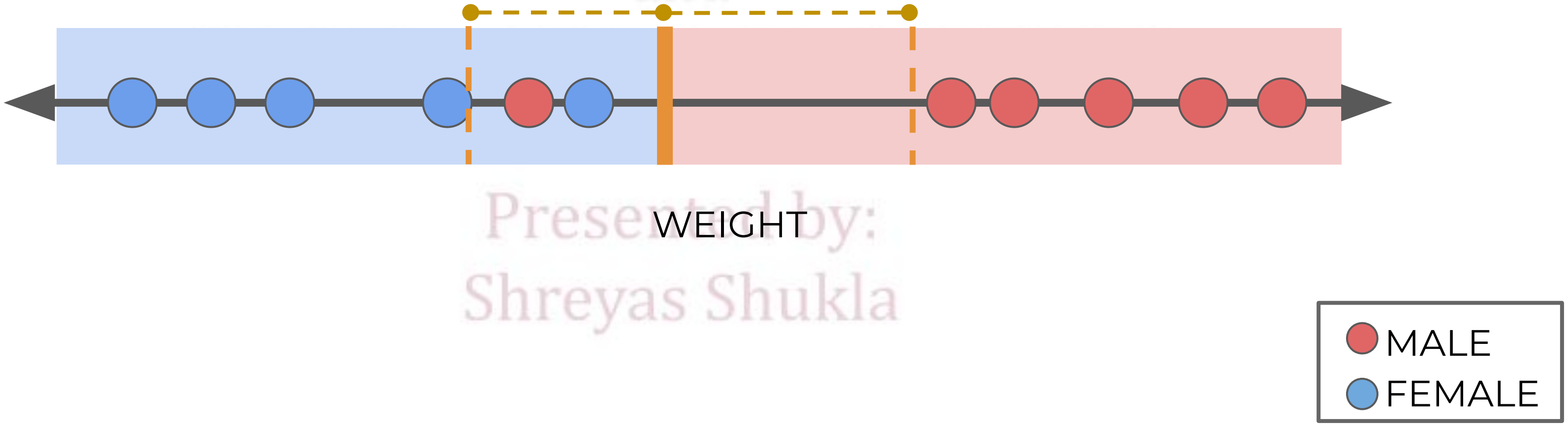
Soft margin: Distance between threshold and the observations



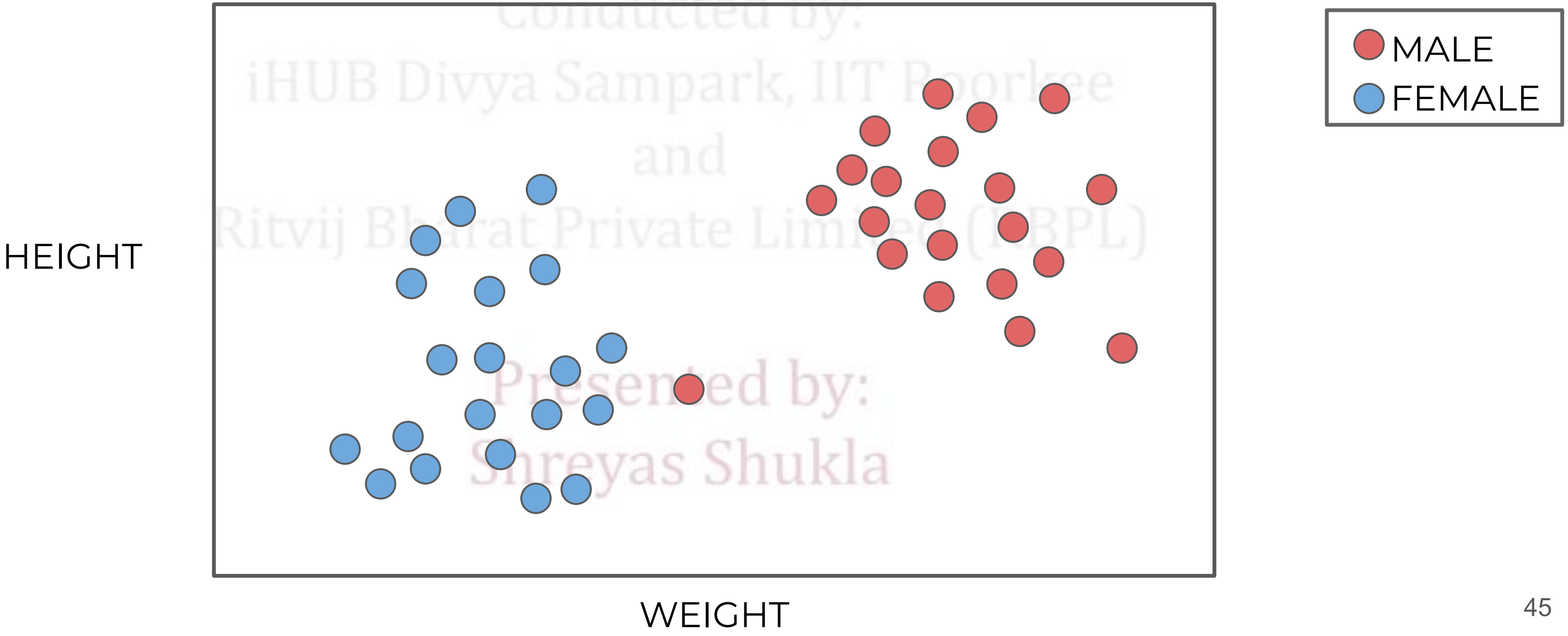
Many threshold splits possible if we allow for soft margins.



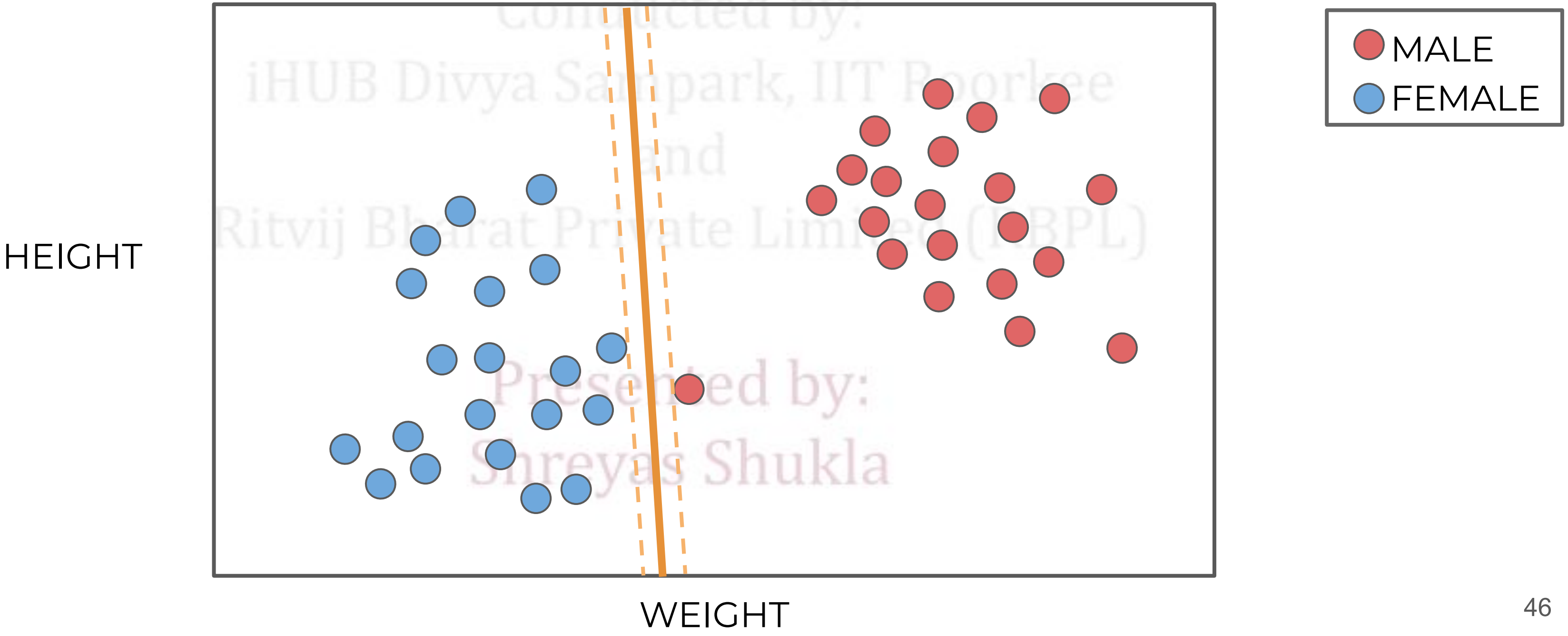
Use cross validation to determine the optimal size of the margins.



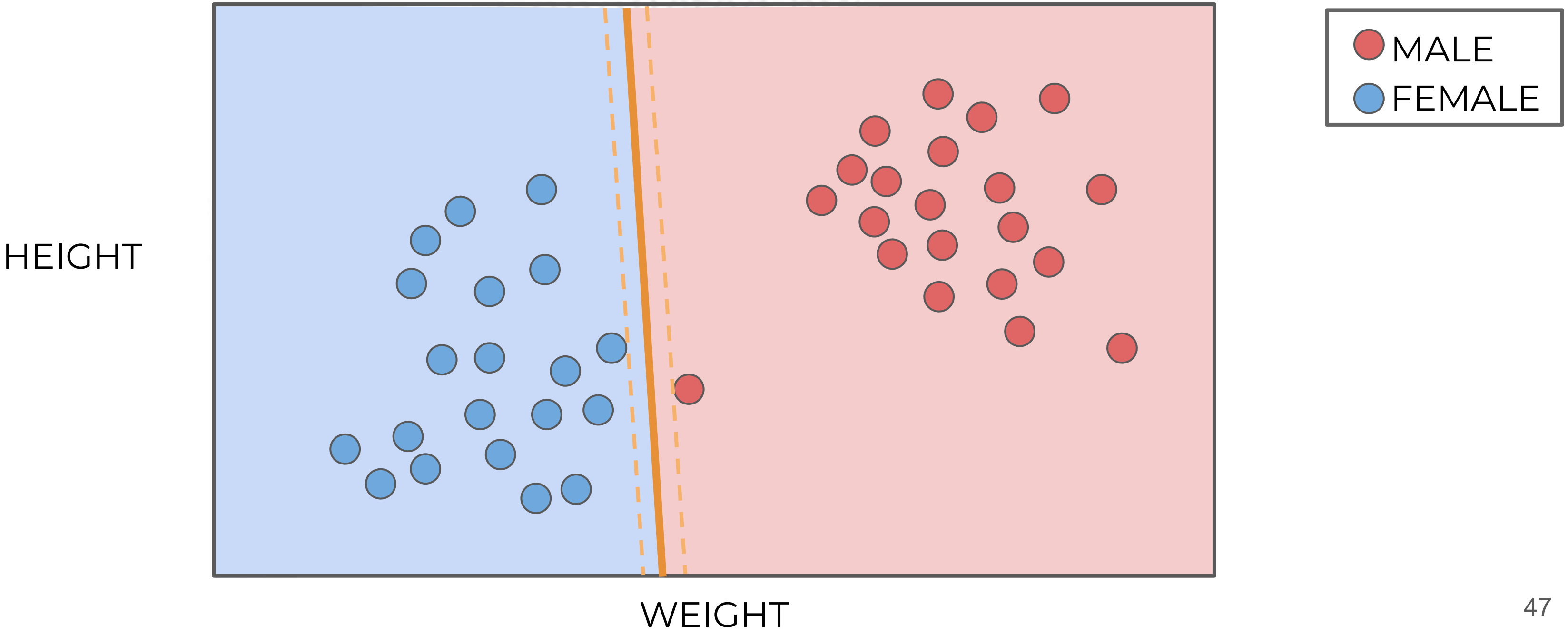
Here, dataset is technically perfectly separable



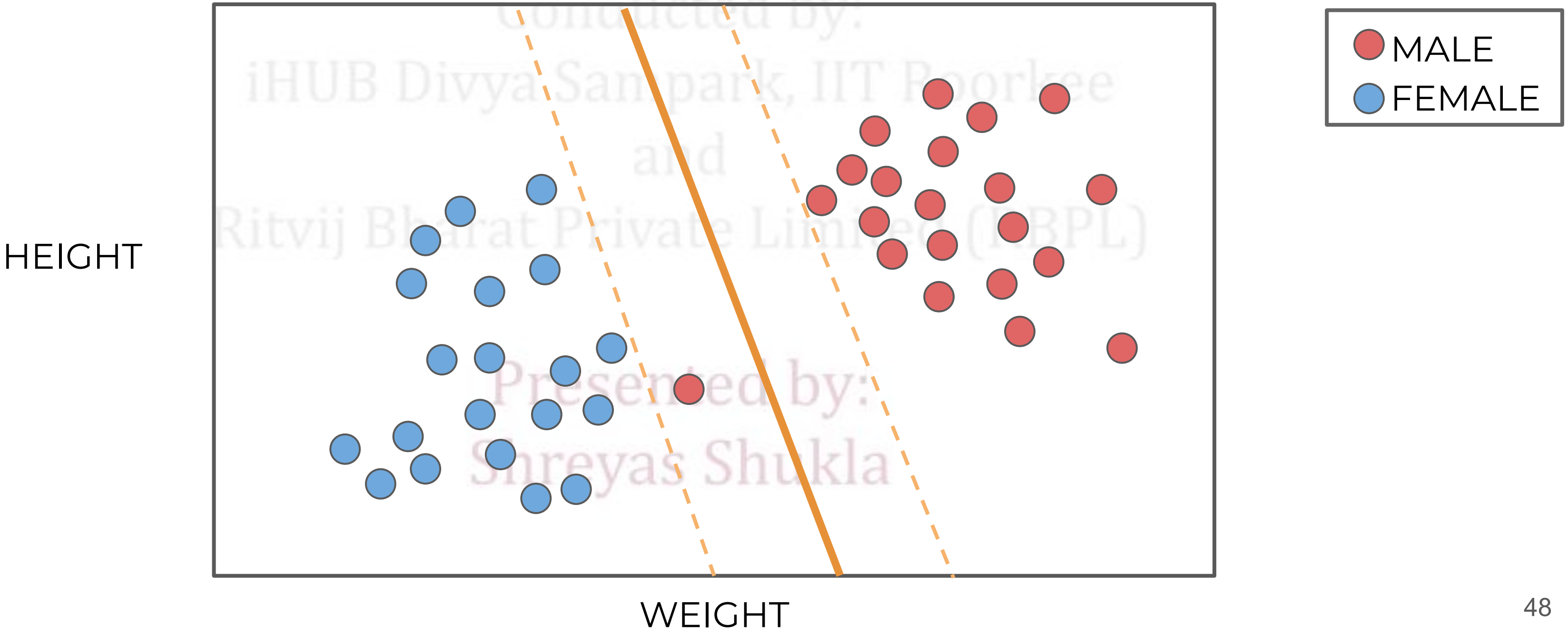
Maximal Margin Classifier



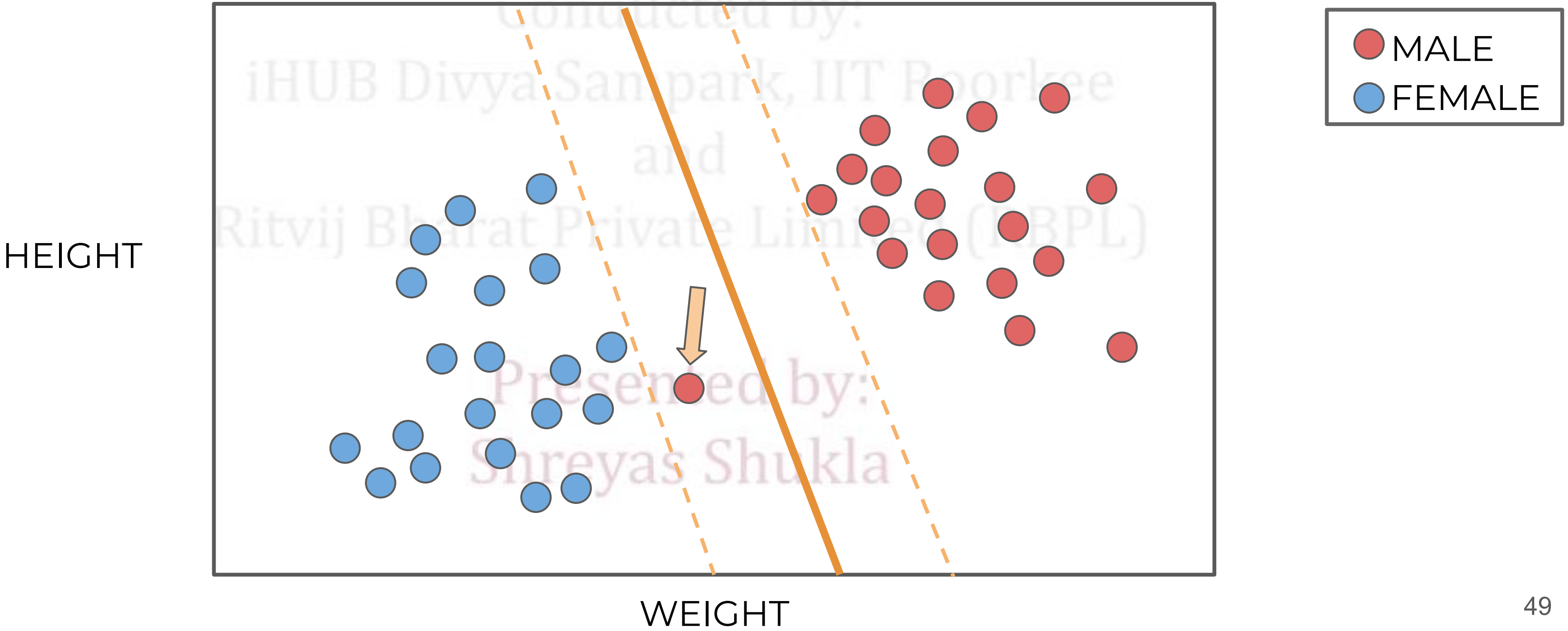
Maximal Margin Classifier



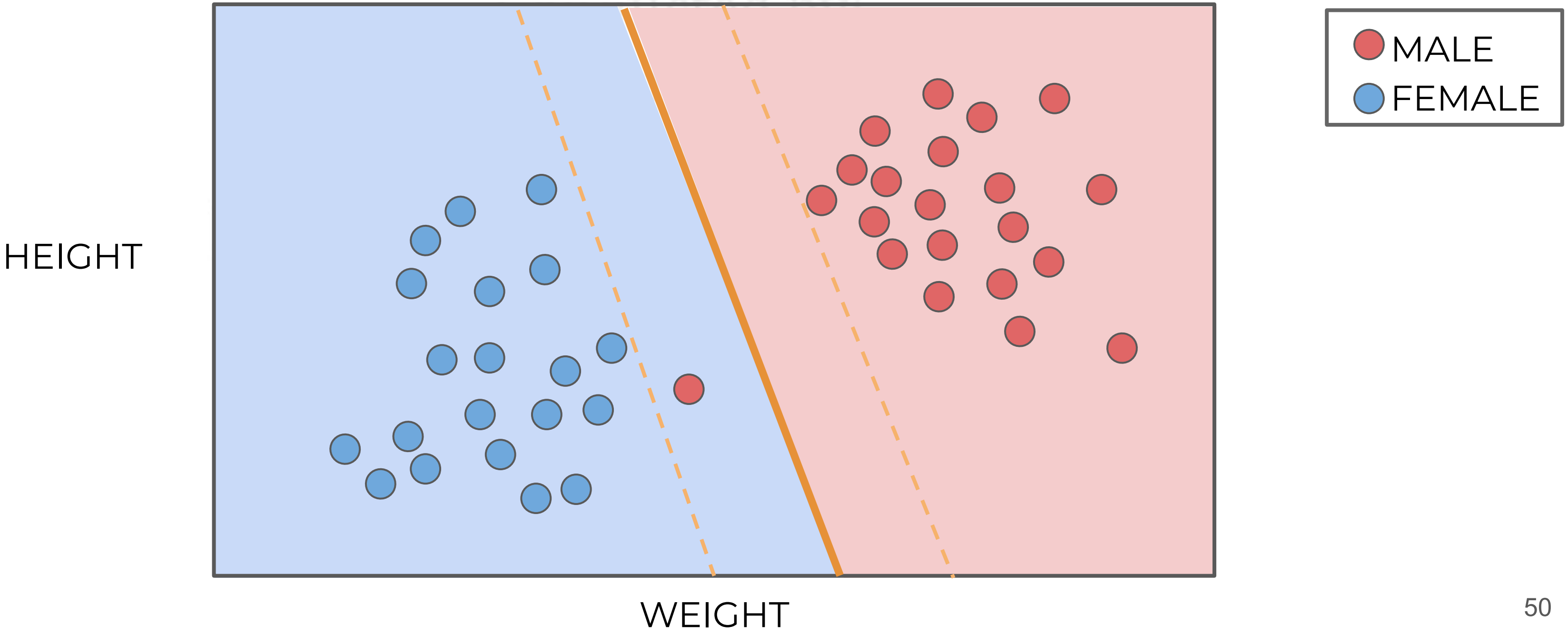
Support Vector Classifier (Soft Margins)



Support Vector Classifier (Soft Margins)



Support Vector Classifier (Soft Margins)

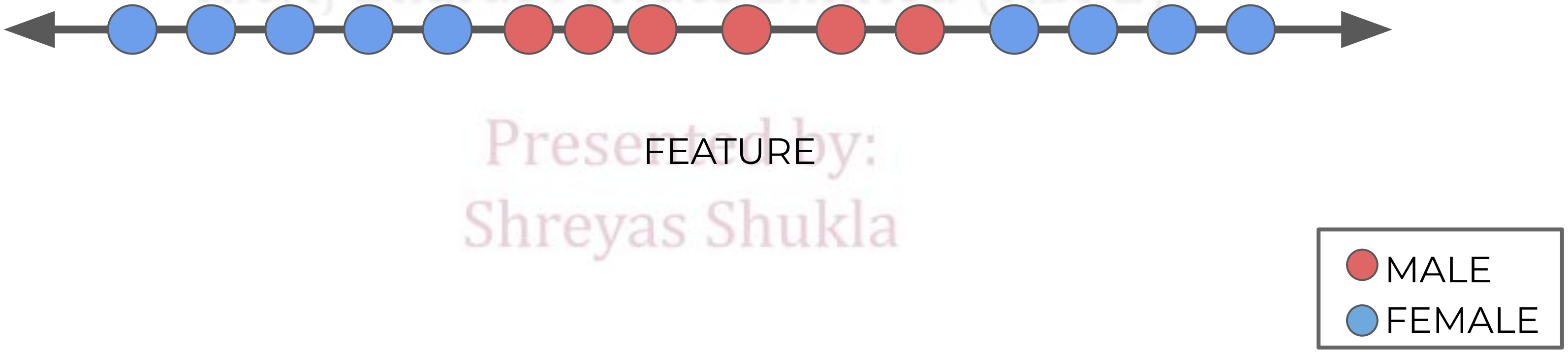


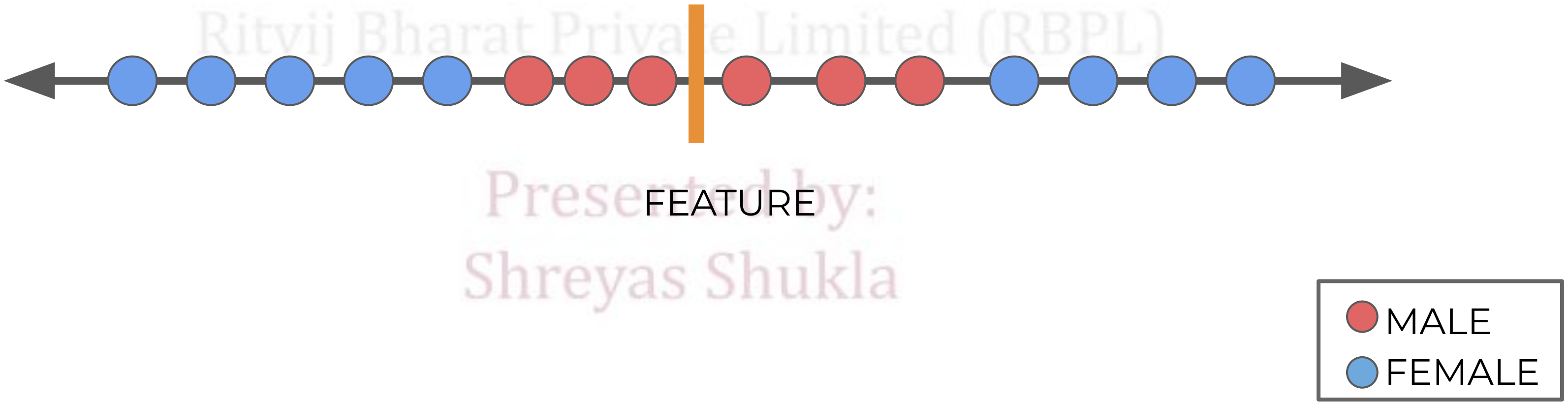
We’ve only visualized cases where the classes are easily separated by the hyperplane in the original feature space.

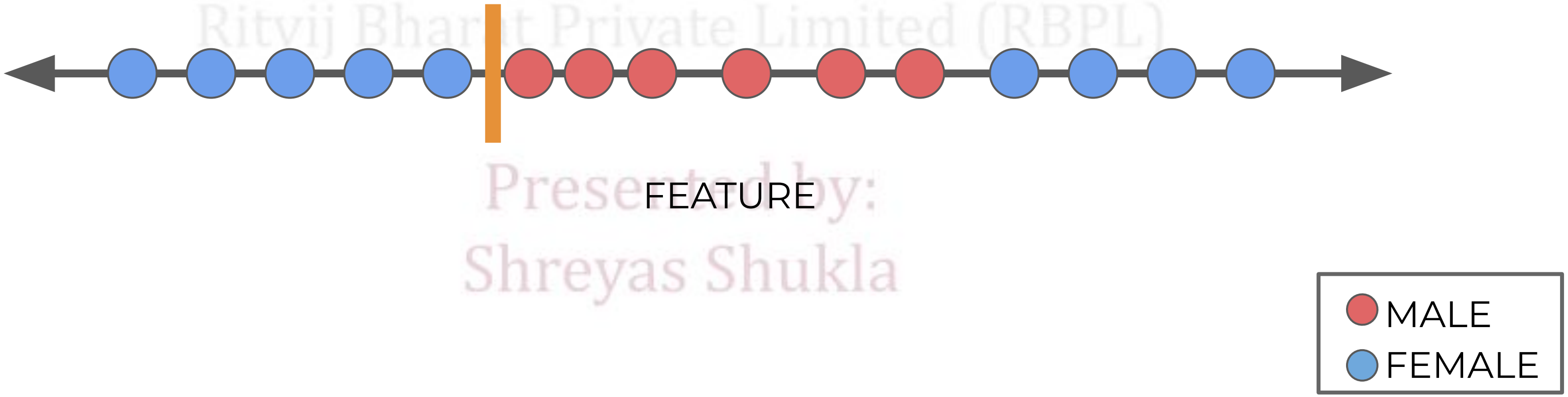
This leaves space for some misclassifications that will still result in reasonable results.

But what if a hyperplane performs poorly, even when allowing for misclassifications?

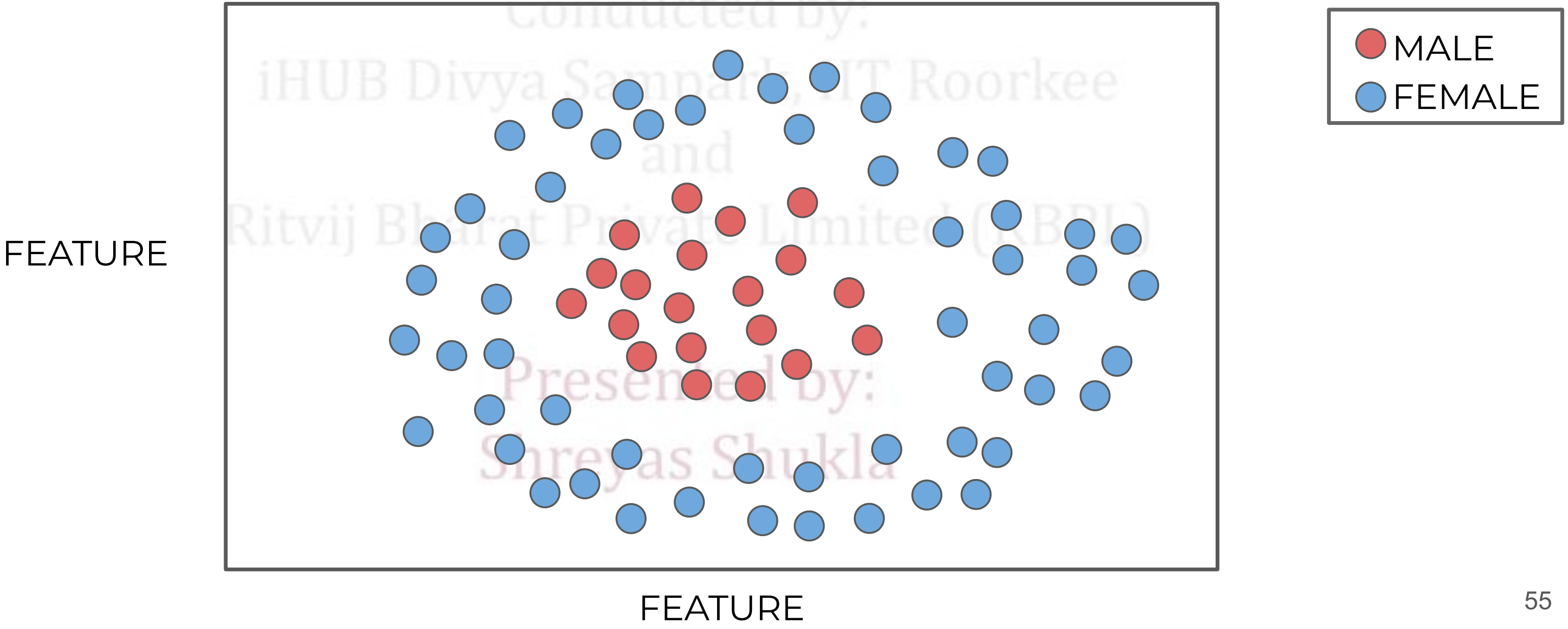
Notice a single hyperplane won’t separate out the classes without many misclassifications!




FEATURE
MALE
FEMALE


FEATURE
MALE
FEMALE

Can’t split classes with hyperplane line:



To solve such cases, we move on from Support Vector Classifier, to Support Vector Machines.

SVMs employ kernels to transform the data into a higher-dimensional space, enabling the utilization of a hyperplane in this elevated dimension for data separation purposes.

An Introduction to Machine Learning with Python Programming
11 Sep 2023 - 20 Oct 2023

Conducted by:
iHUB Divya Sampark, IIT Roorkee
and
Ritvij Bharat Private Limited (RBPL)

Support Vector Machines

Theory and Intuition - Kernels

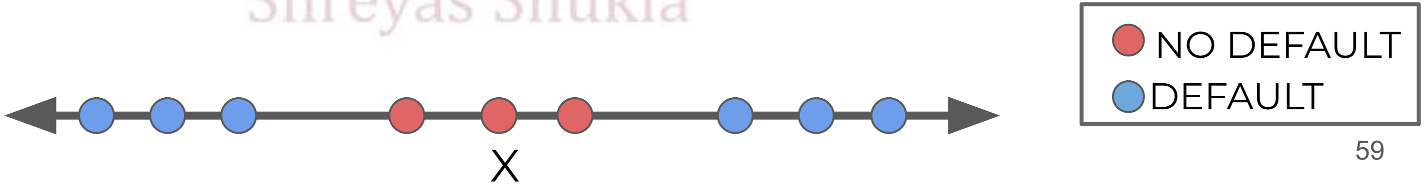
Presented by:
Shreyas Shukla

In Kernels, we move beyond a Support Vector Classifier and use Support Vector Machines.

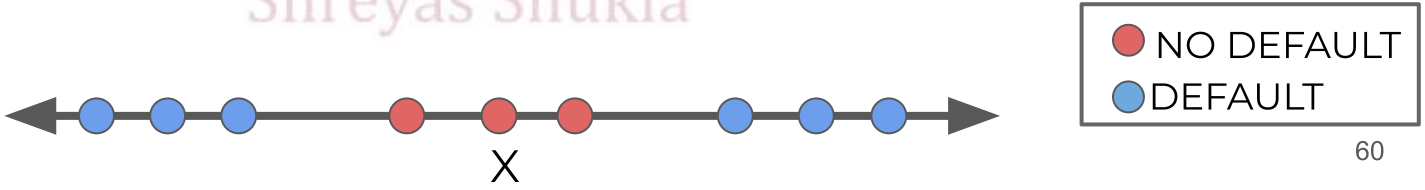
Variety of kernels can be used to “project” the features to a higher dimension.

Let’s see how this works

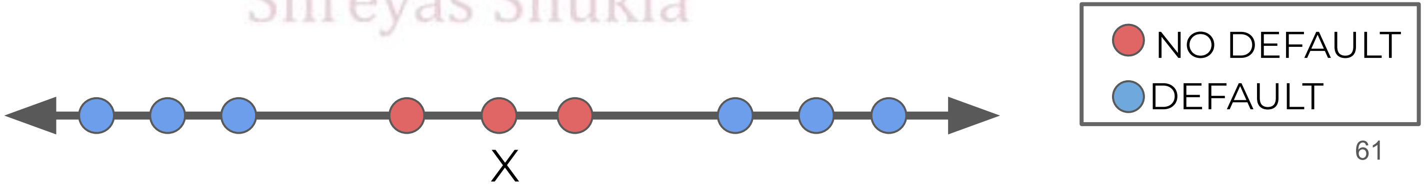
Recall our 1D example where classes were not easily separated by a single hyperplane:



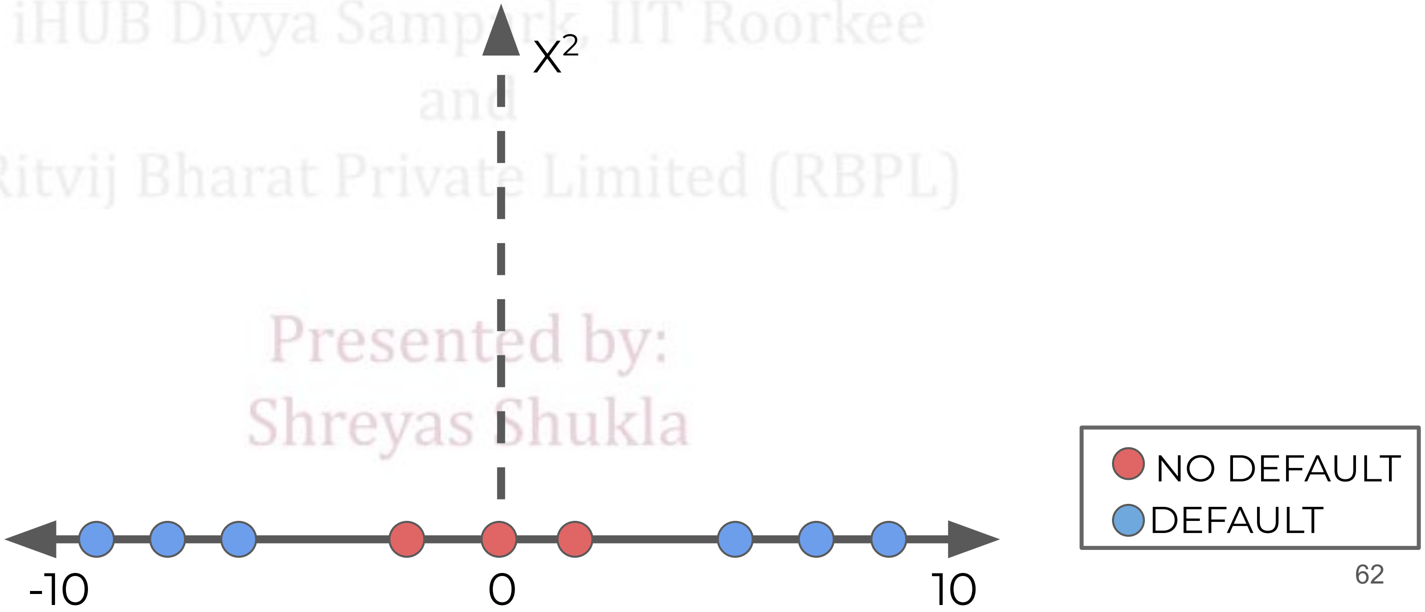
Let’s explore how using a kernel could project this feature onto another dimension.



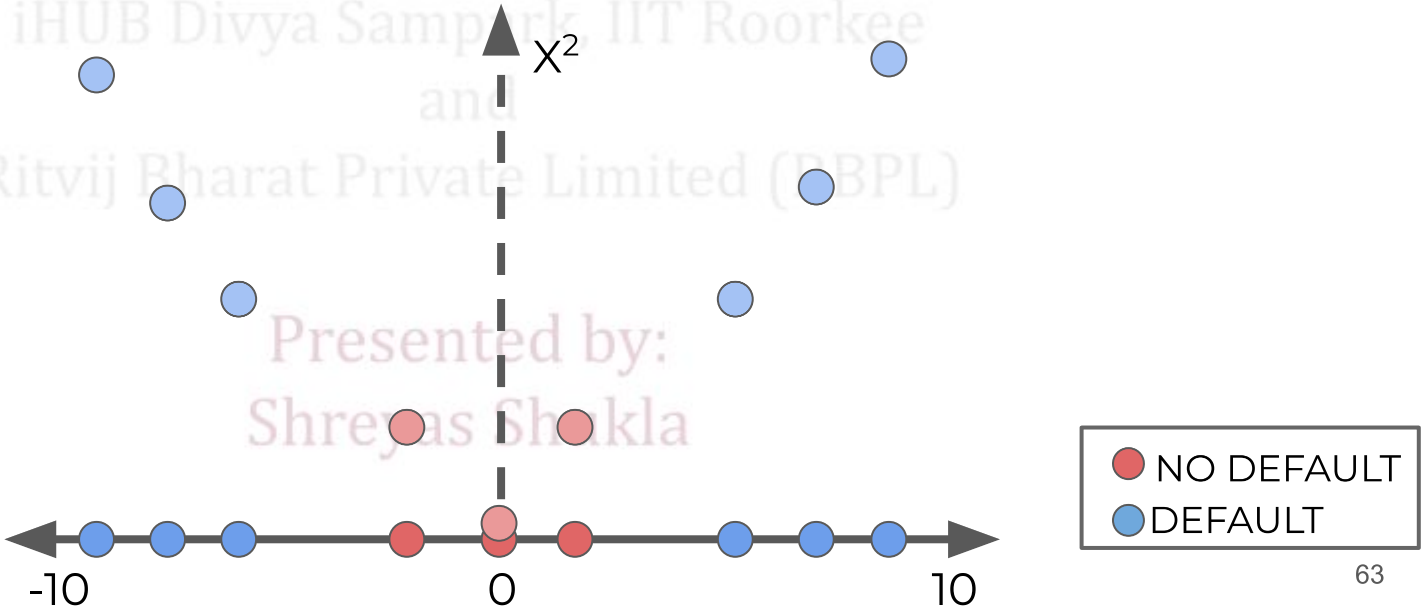
For example, a polynomial kernel could expand onto an X^2 dimension:



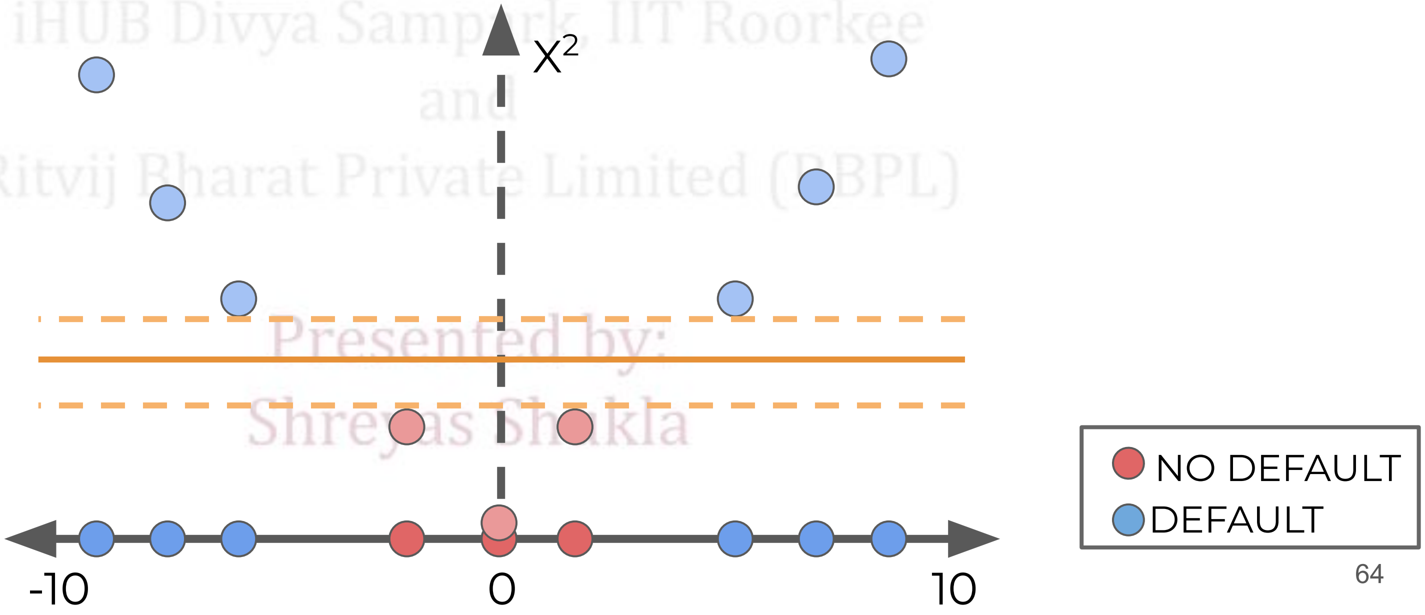
For example, a polynomial kernel could expand onto an X^2 dimension:



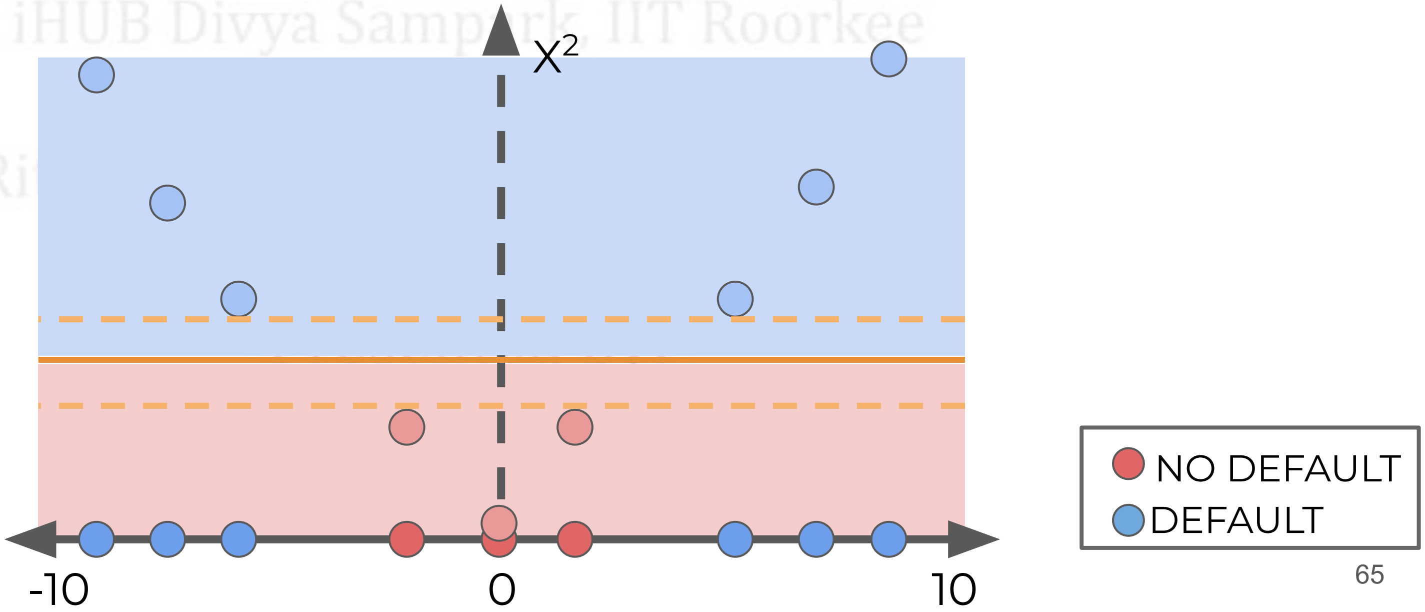
For example, a polynomial kernel could expand onto an X^2 dimension:



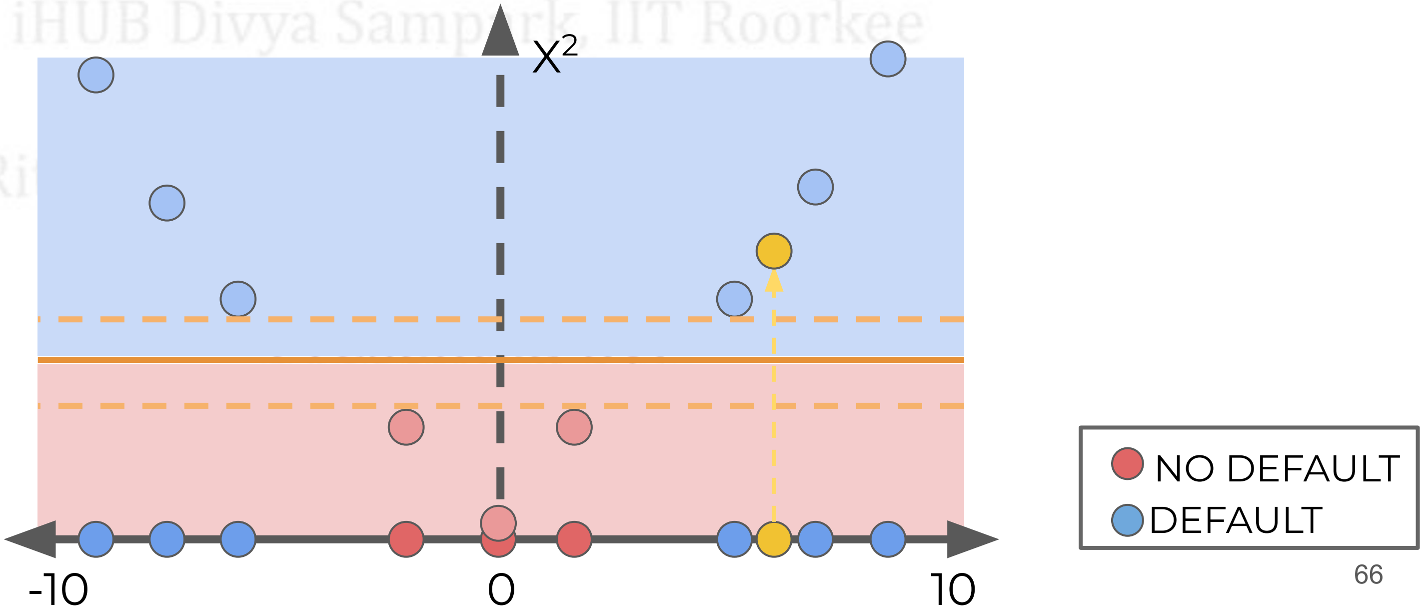
Create a hyperplane after projecting



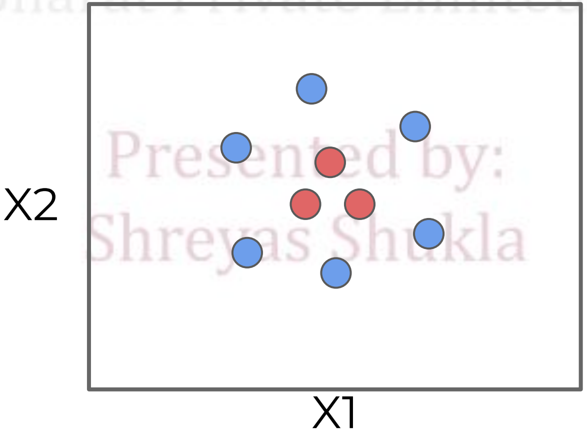
Create a hyperplane after this projection. Using this kernel projection, evaluate new points:



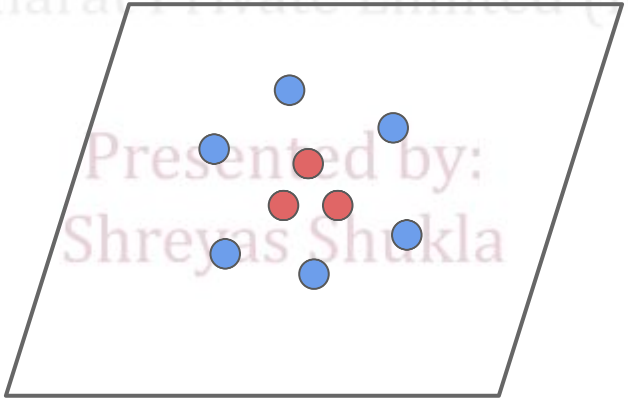
Evaluate new points



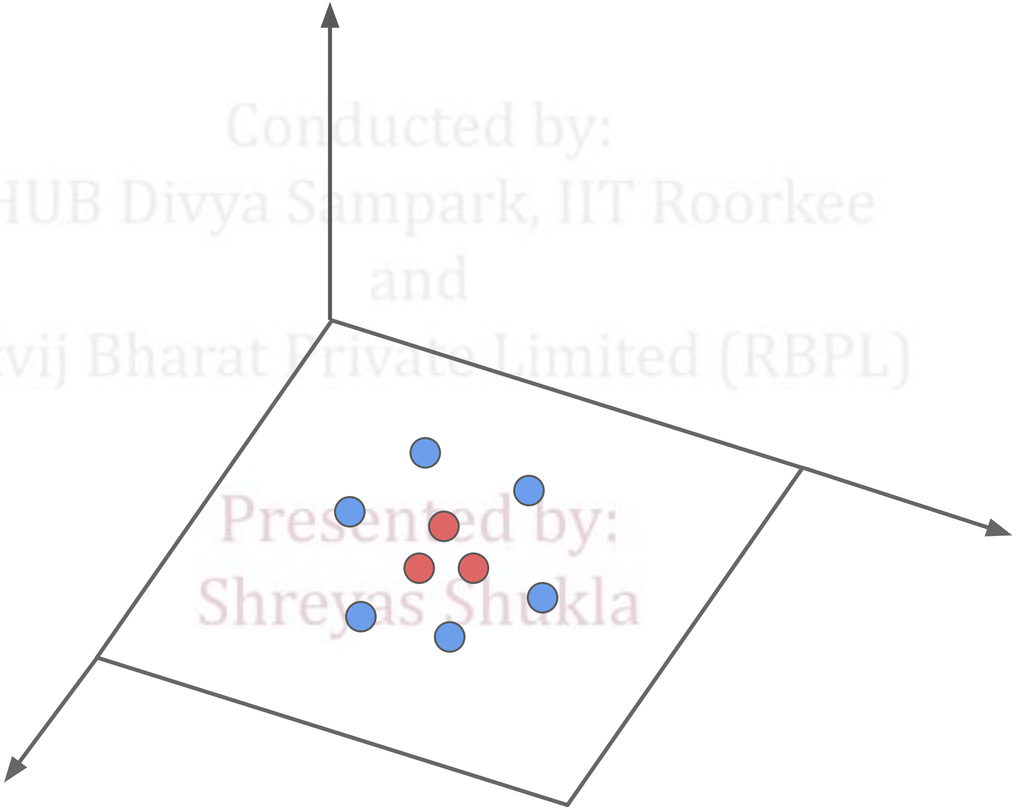
Now Imagine a 2D feature space where a hyperplane can not separate effectively, even with soft margins.



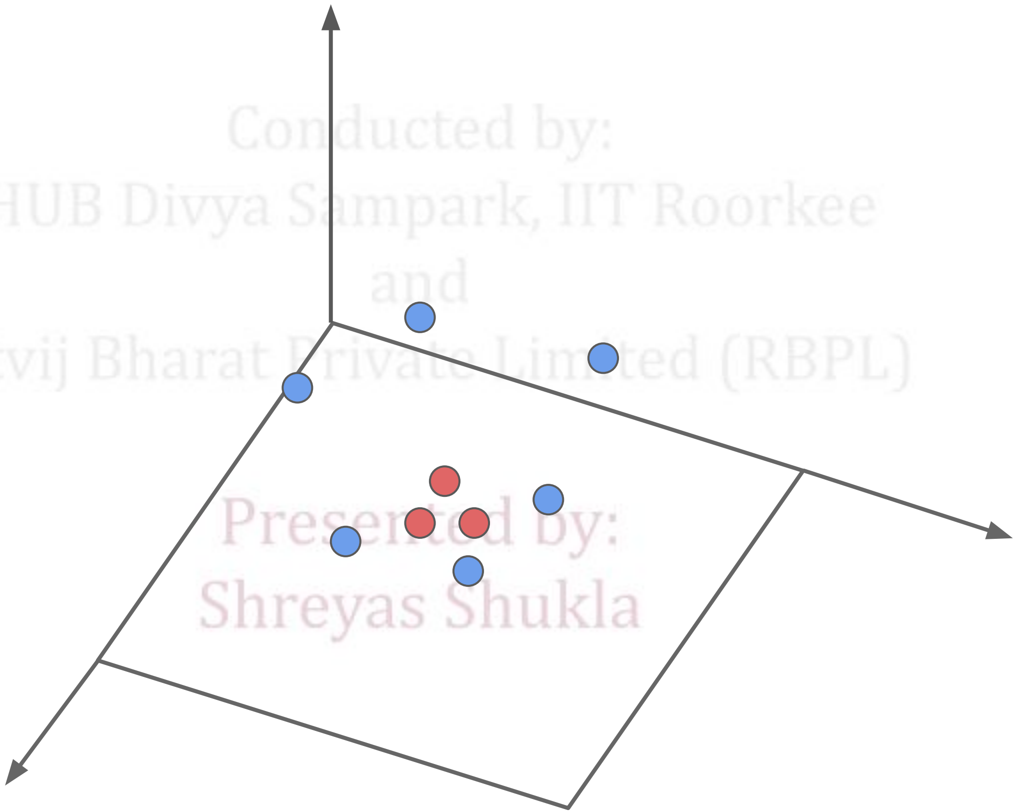
Here, we use SVMs to enable the use of a kernel transformation to project to a higher dimension.



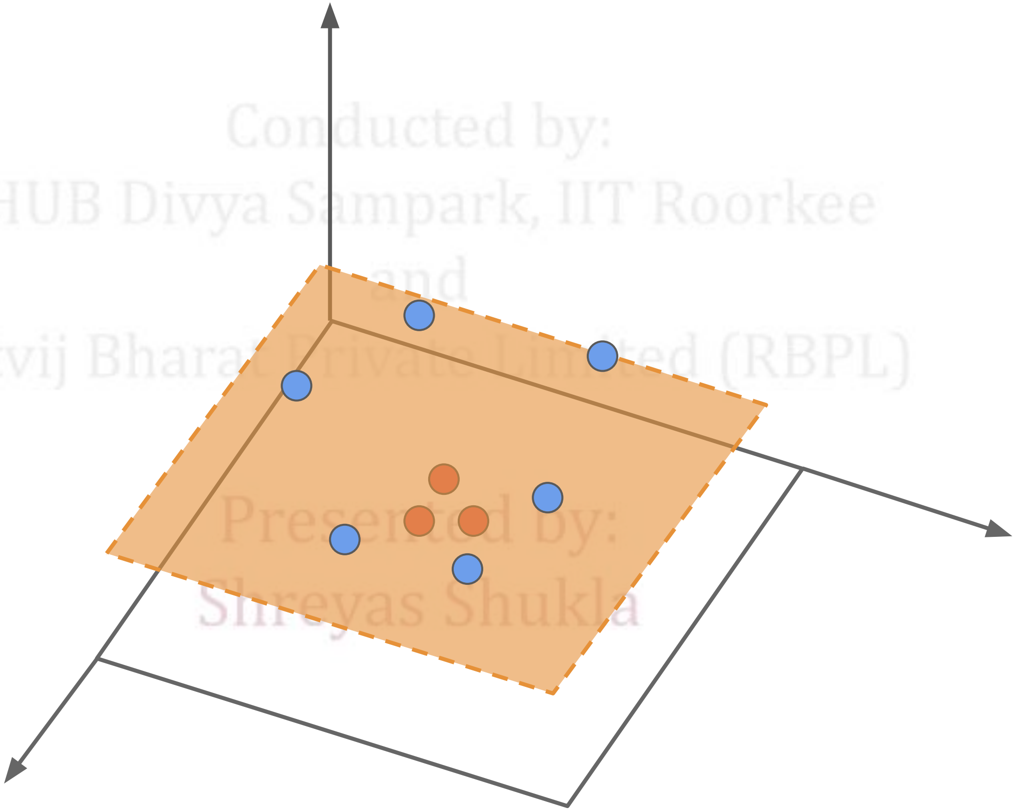
2D to 3D



2D to 3D



Hyperplane



Using kernels in SVM is “**kernel trick**”.

We already 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!

How does the kernel trick achieve this task?

It takes advantage of dot products of the transpositions of the data that we shall see in the next lecture

We will go through the basic mathematical ideas behind the “kernel trick” (Optional, feel free to avoid)!

Support Vector Machines

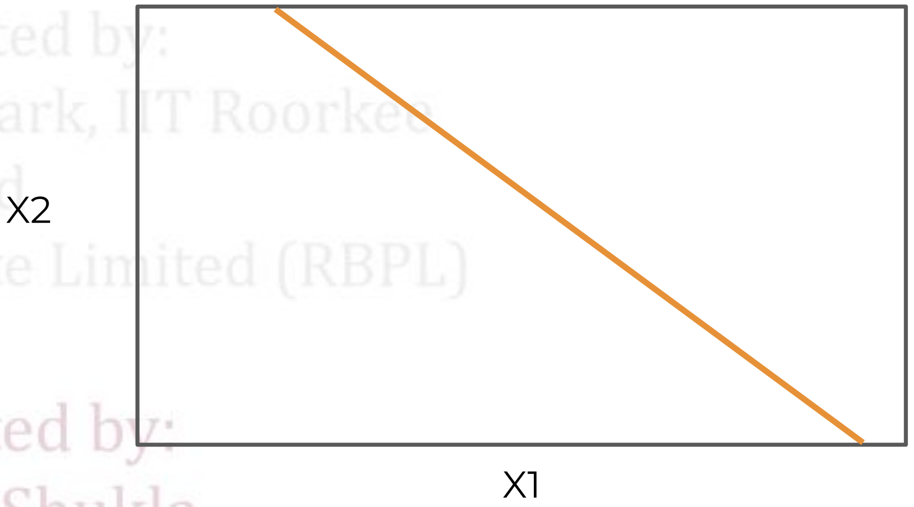
Theory and Intuition - Kernel Trick and Math

Let's briefly talk about general mathematics of SVM and how it is related to the Scikit-Learn class calls.

Feel free to consider this an "optional" lecture.

Hyperplanes Defined

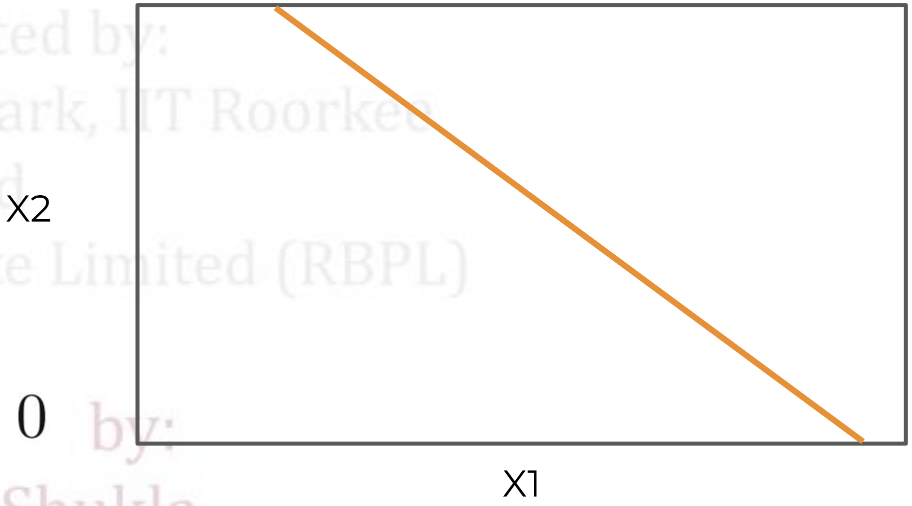
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$



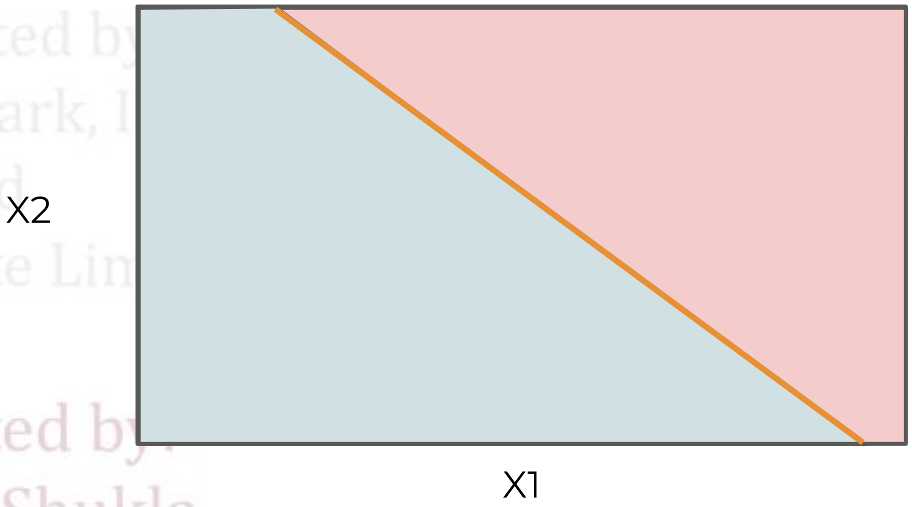
Hyperplanes Defined

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

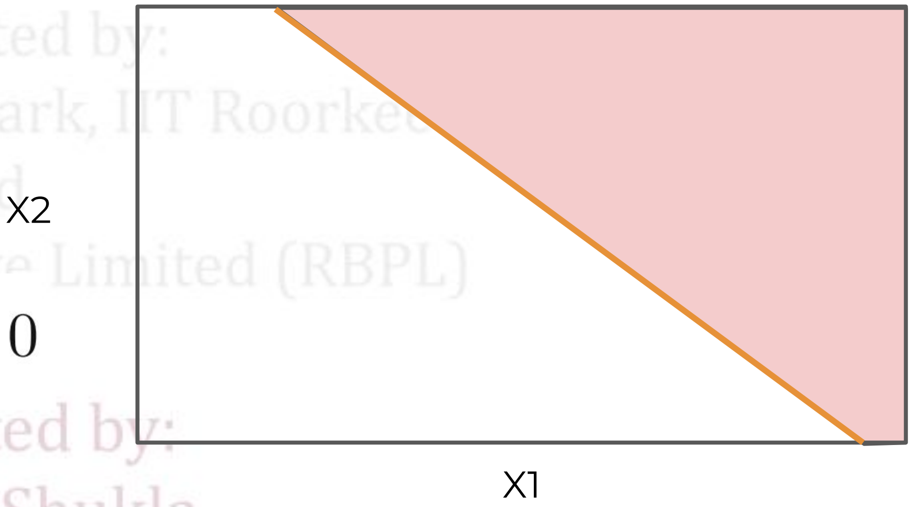
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p = 0$$



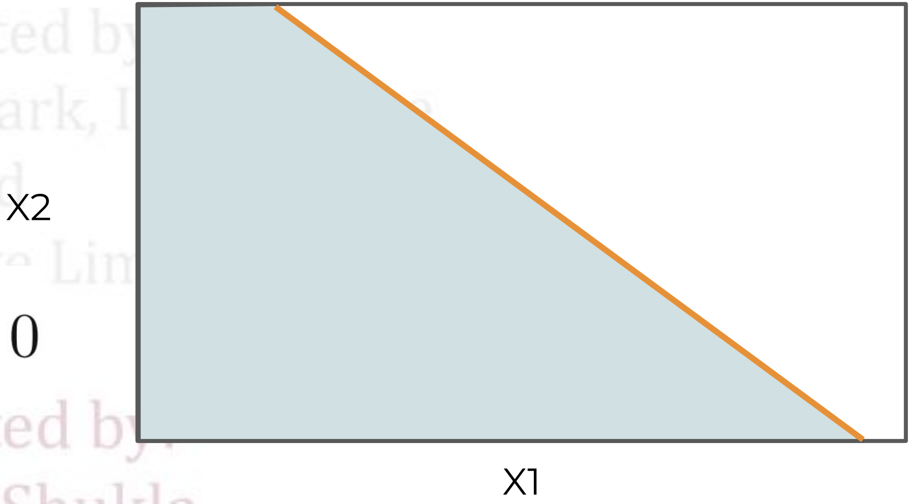
Separating Hyperplanes



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



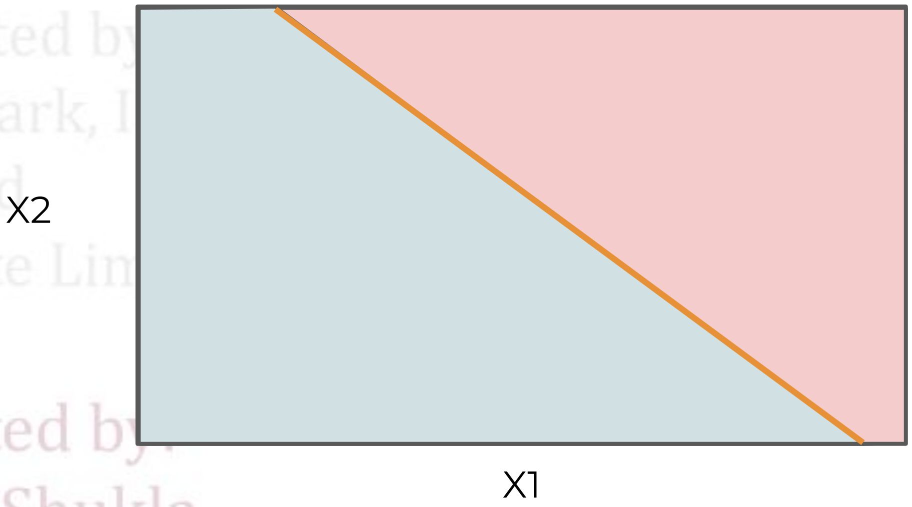
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p < 0$$



Separating Hyperplanes

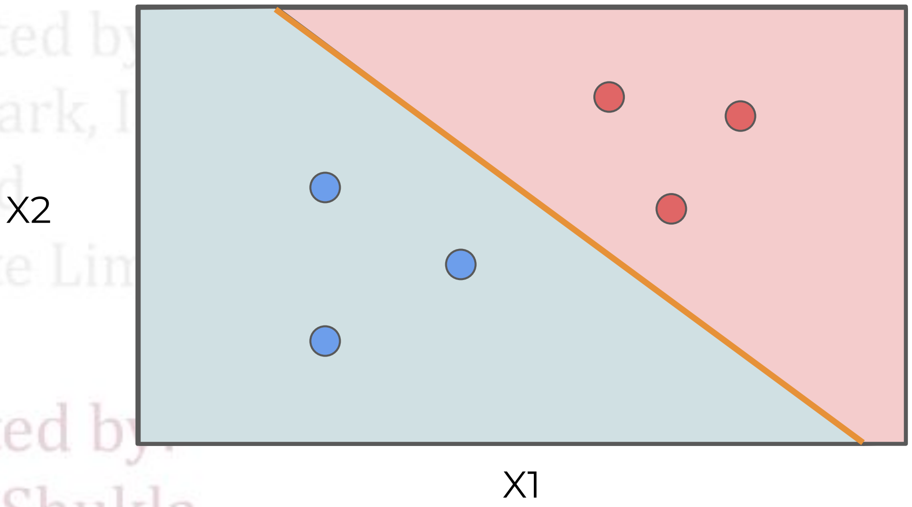
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p > 0$$

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



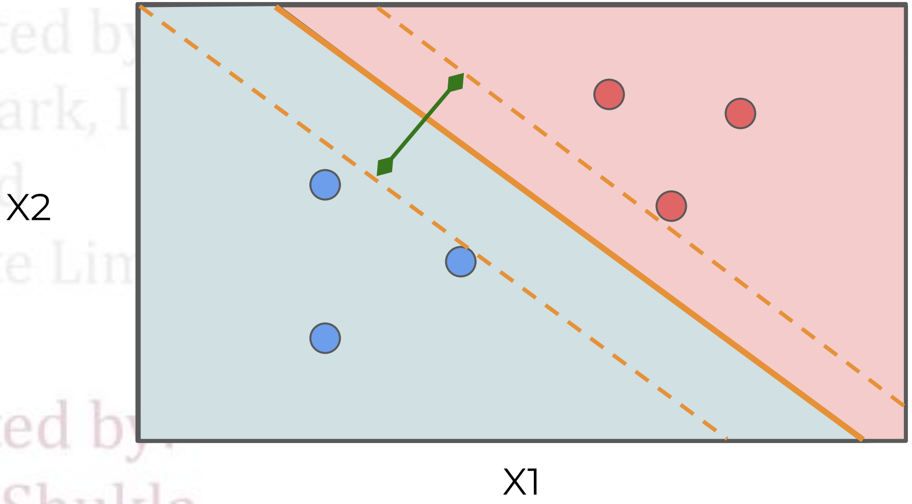
Data Points

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



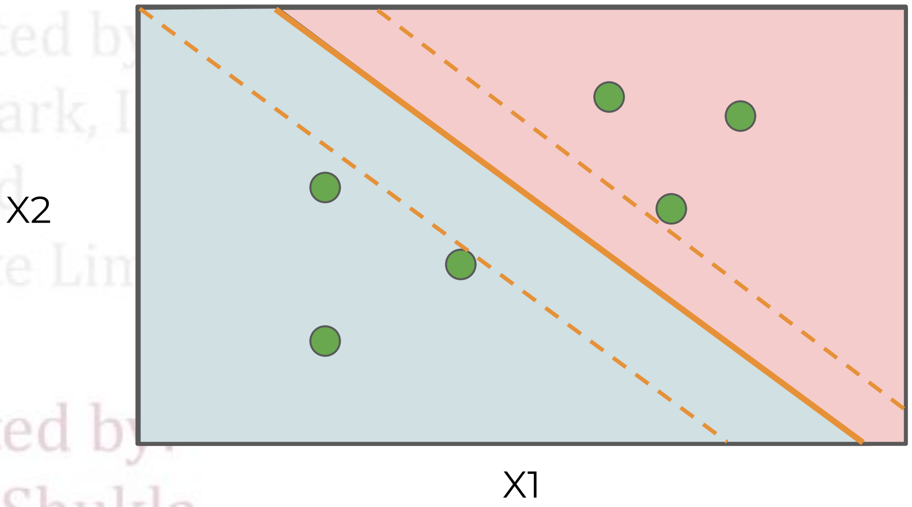
Max Margin Classifier

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



Max Margin Classifier

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



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

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

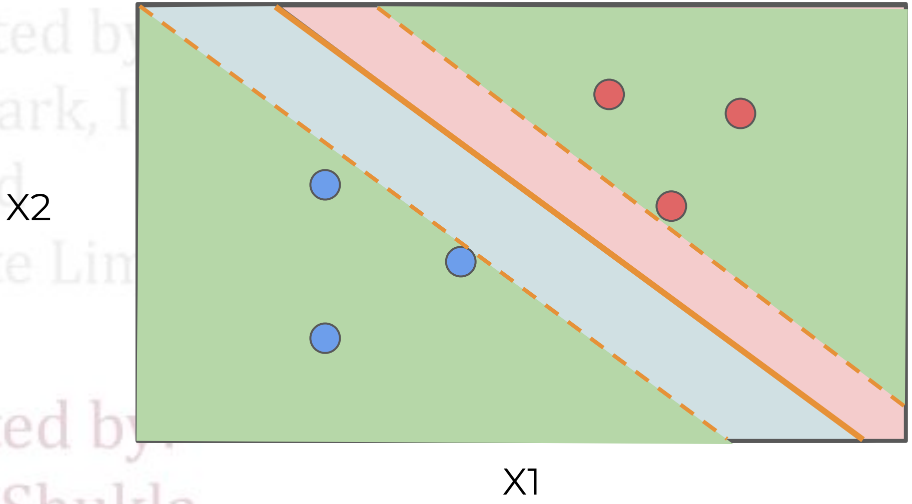
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip}) \geq M \quad \forall\, i = 1, \ldots, n.$$

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

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

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

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

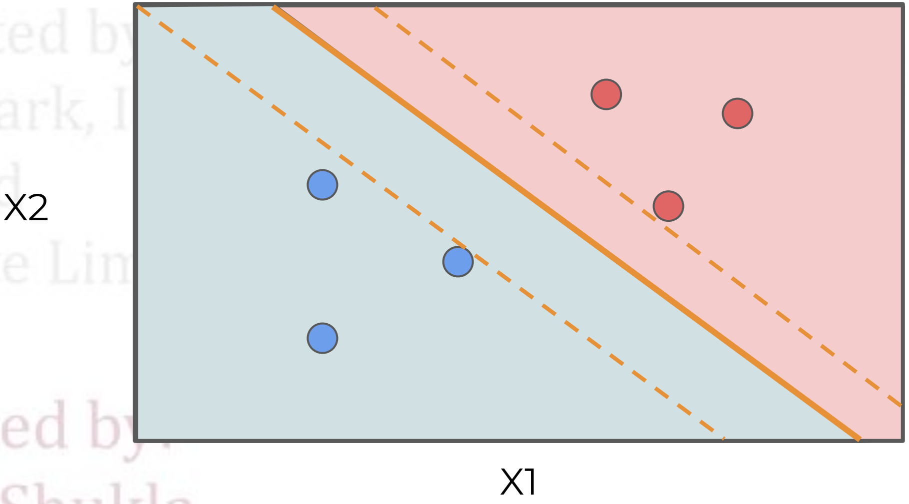


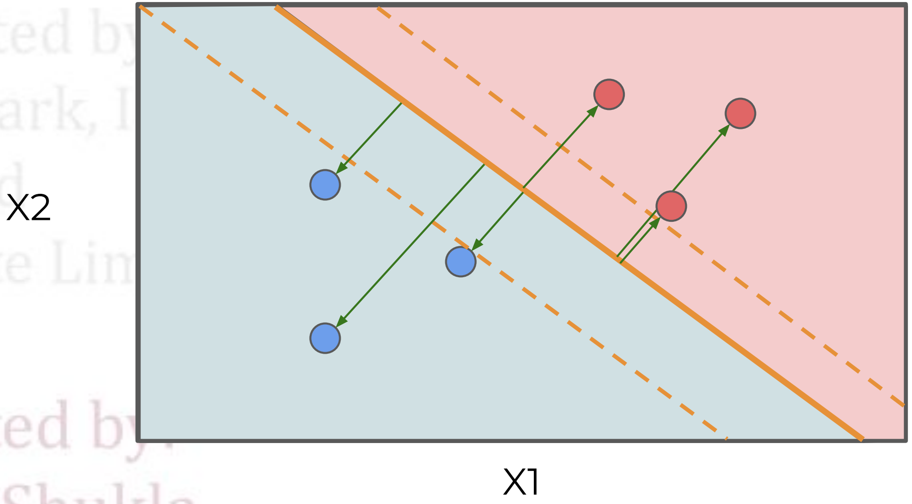
$$x_1 = \begin{pmatrix} x_{11} \\ \vdots \\ x_{1p} \end{pmatrix}, \ldots, x_n = \begin{pmatrix} x_{n1} \\ \vdots \\ x_{np} \end{pmatrix}$$

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

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

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





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

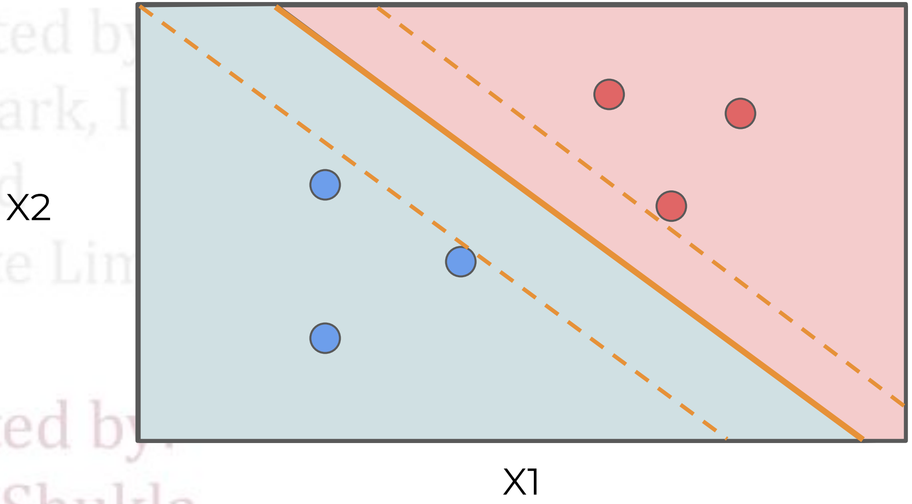
$$y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip})$$

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

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

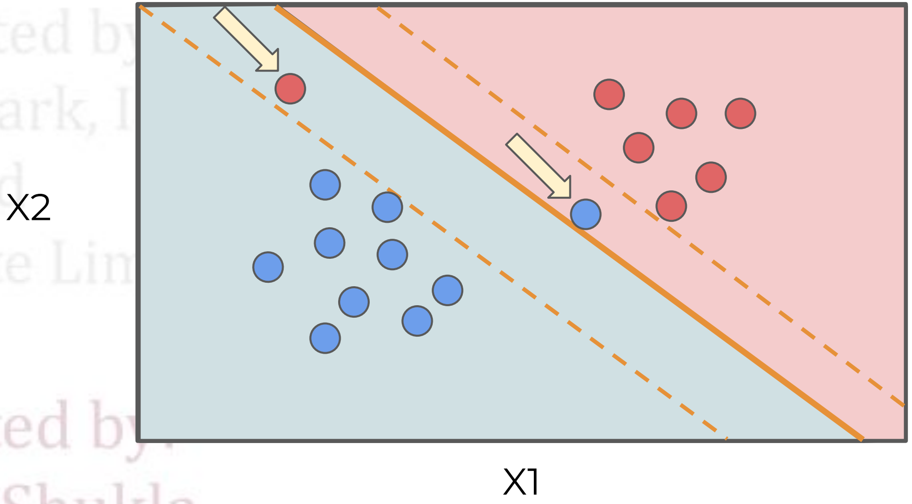
$$\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1$$

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



Support Vector Classifier

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



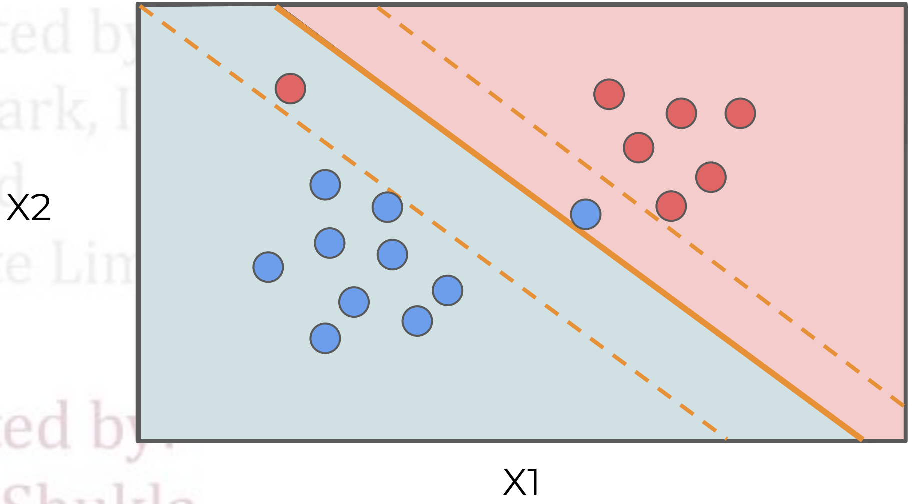
Support Vector Classifier

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

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

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

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



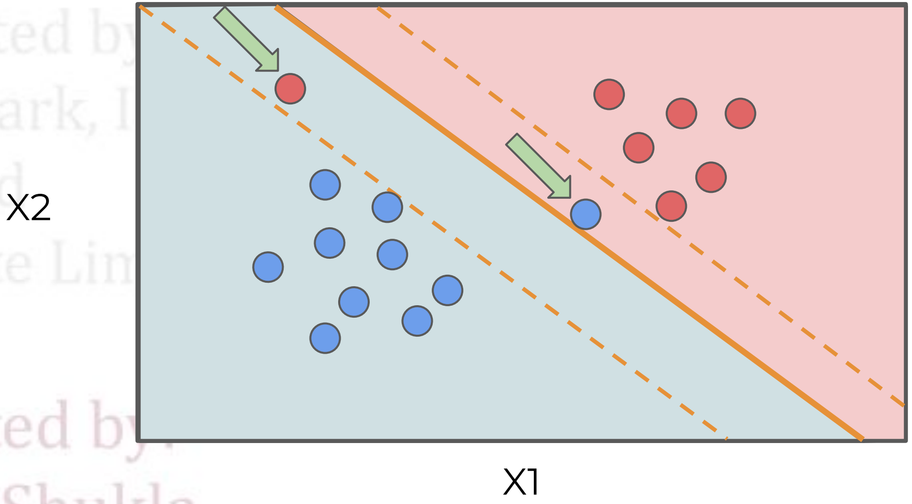
Support Vector Classifier

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

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

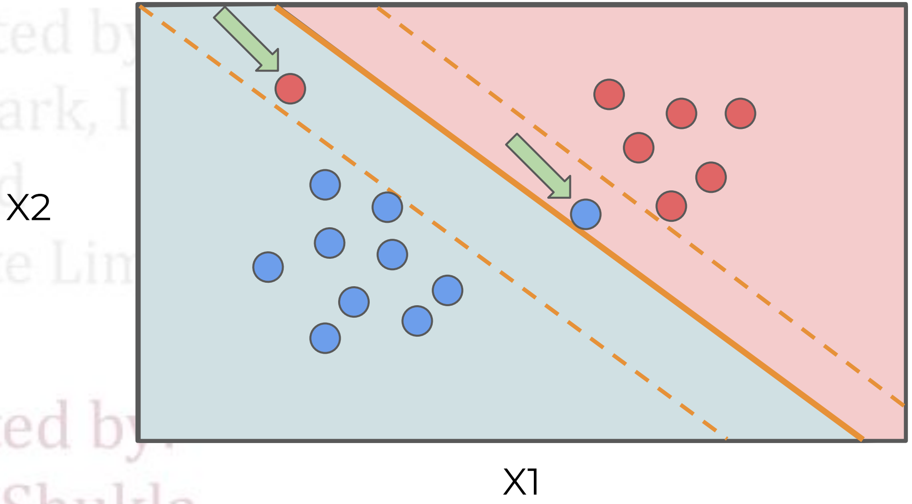
$$\boxed{\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C}$$

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



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.

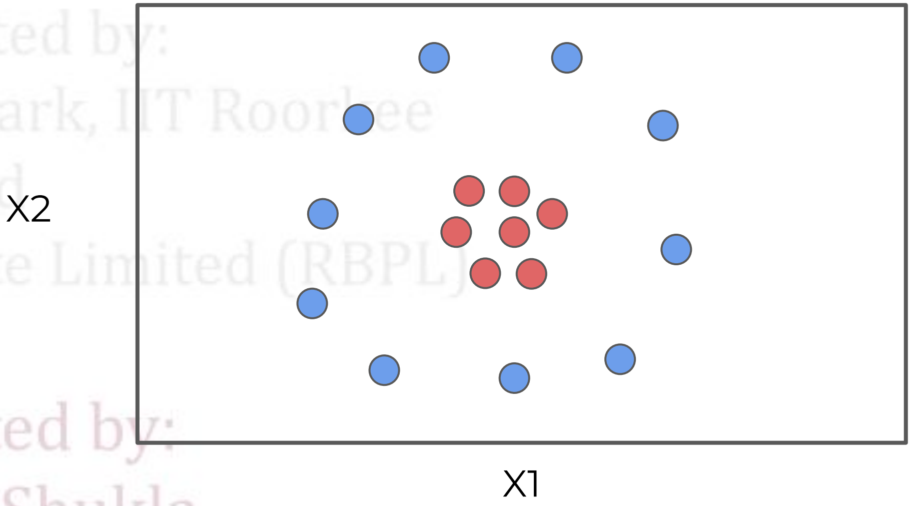


$$\epsilon_i \geq 0, \quad \sum_{i=1}^{n} \epsilon_i \leq C$$

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

Support Vector Machines

$$X_1, X_2, \ldots, X_p,$$

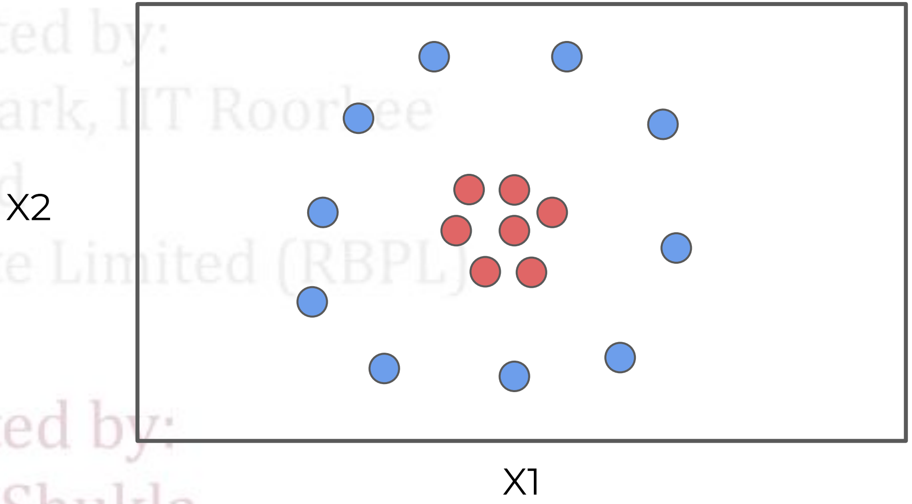


Support Vector Machines

$$X_1, X_2, \ldots, X_p,$$

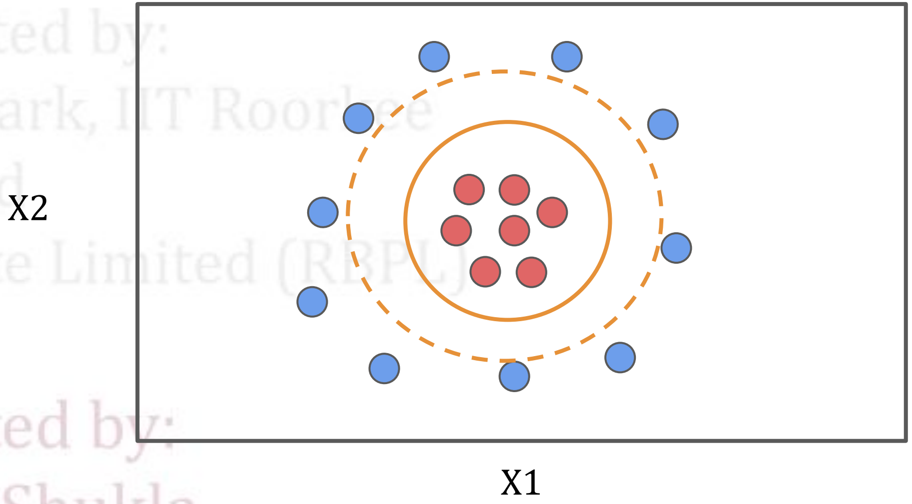
$$\Downarrow$$

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



Support Vector Machines

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



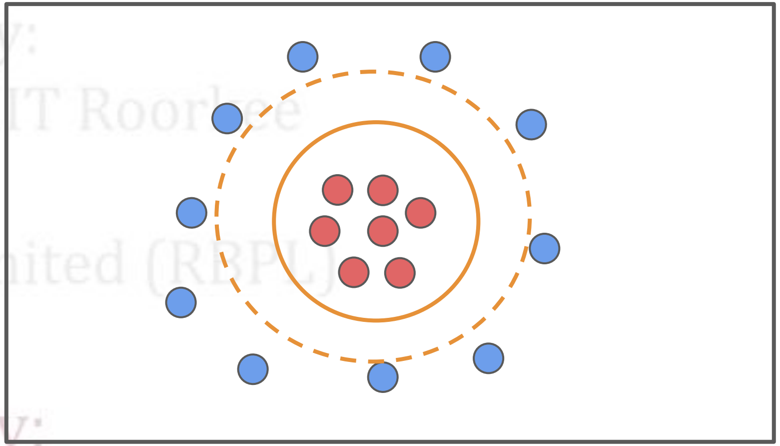
Support Vector Machines

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

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

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

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



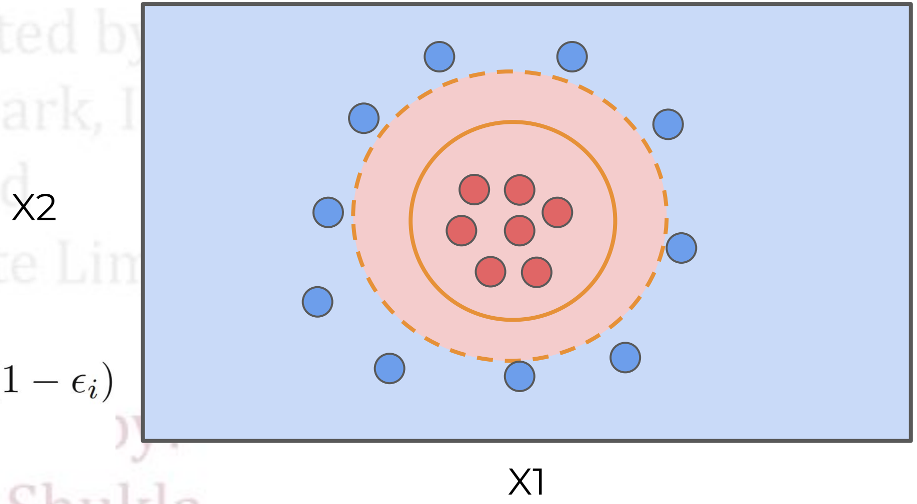
Support Vector Machines

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

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

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

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

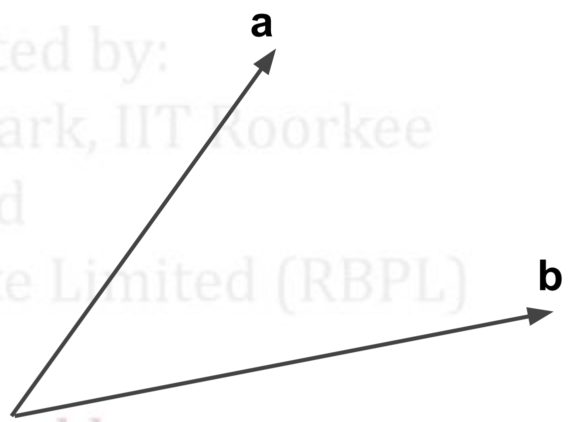


How to deal with very large feature space? As polynomial order growS, the number of computations necessary to solve for margins also grows!

We use **Kernel trick** which makes use of the **inner product** of vectors, also known as the **dot product**.

Dot Product

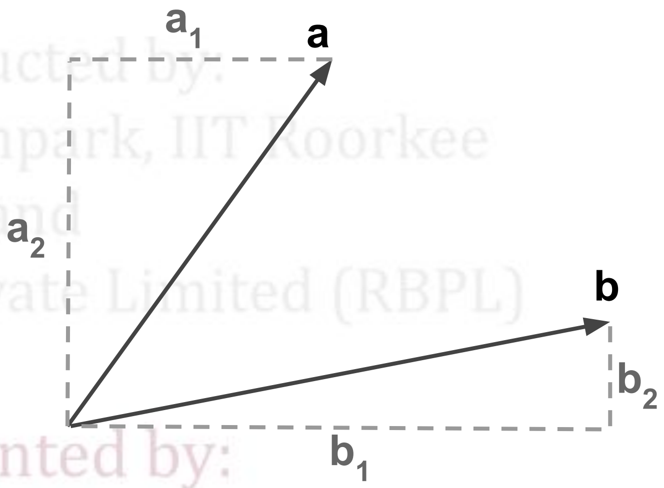
$$\langle a, b \rangle = \sum_{i=1}^{r} a_i b_i$$



Dot Product

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

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

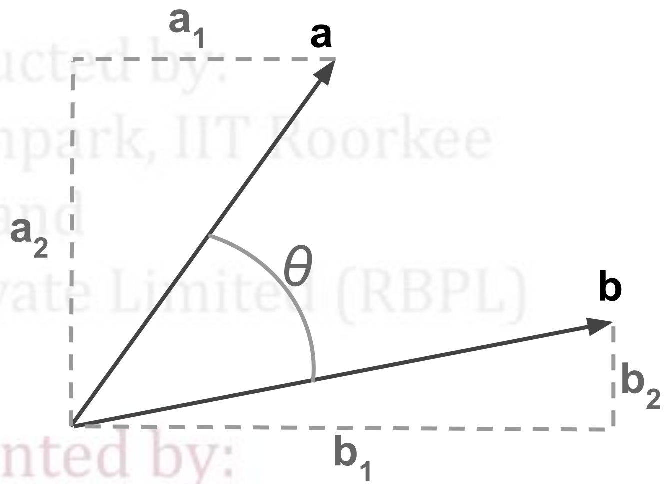


Dot Product

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

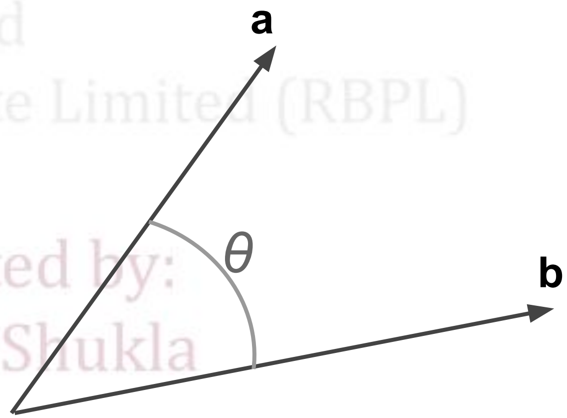
$$a \cdot b = a_1 b_1 + a_2 b_2$$

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



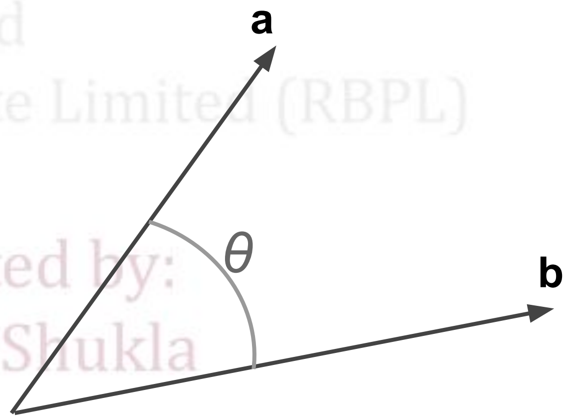
Notice how the dot product can be thought of as a similarity between the vectors.

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



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

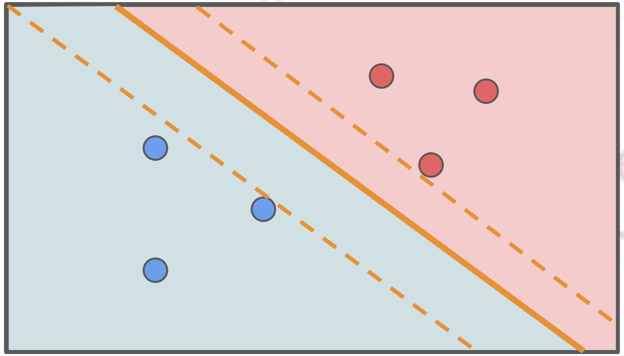
$$a \cdot b = |a||b|cos(\theta)$$



Let's discuss Kernel Trick

Linear Support Vector Classifier rewritten:

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

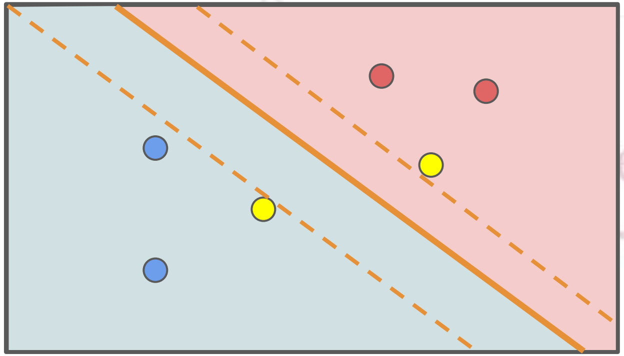


Calculating the inner products of all pairs of training observations

Linear Support Vector Classifier rewritten:

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

Only non-zero for the support vectors.



Linear Support Vector Classifier rewritten:

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

$$\Downarrow$$

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

Linear Support Vector Classifier rewritten:

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

S collection of indices of these support points

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.

Kernel Function

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

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

Kernel Function

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

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

Kernel Function

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

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

Polynomial Kernel

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

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

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

Radial Basis Kernel

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

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

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

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.