Feature Engineering

In the real world, not every data set is machine learning ready

we often need to perform data cleaning or try to produce more usable features.

Feature Engineering is the process of using domain knowledge to extract features from raw data via data mining techniques.

But what does this actually mean?

Three general approaches:

- Extracting Information
- Combining Information
- Transforming Information

Extracting Information

- Imagine a dataset with visitor expenditure information for a bar.
- We have a timestamp for each row:
  - 1990-12-01 09:26:03
- In its current format, its very difficult to pass into a machine learning algorithm.

In its current format, its very difficult to pass into a machine learning algorithm.

There is no coefficient we can apply for a non-numeric data point:

- 1990-12-01 09:26:03

In general for most algorithms we need to make sure features are float or int.

Instead we extract information

- 1990-12-01 09:26:03
  - Year: 1990
  - Month: 12
  - Weekday or Weekend (0/1)
  - Mon:1,Tues:2, … Sun:7

More complex examples:

- Text data for deed of house
  - Length of text
  - Number of times certain terms are mentioned

Combining Information

- We’ve actually already done this with Polynomial Regression!
- Recall advertising spend could have possible interaction terms to consider, so we could multiply them together.

- Combining Information
    - Could also combine extracted information:
        - New Feature:
            - 0 or 1 value indicating:
                - Both weekend and evening?

Transforming Information

- Very common for string data
- Most algorithms can not accept string data (can’t multiply a string such as “red” by a numeric coefficient)

Mastering Machine Learning with Python
(27th Aug 2024 - 18th Oct 2024)
iHUB DivyaSampark, IIT Roorkee
and
Ritvij Bharat Private Limited

Transforming Information

- Often categorical data is presented as string data.
- For example a large data set of social network users could have country of origin as a string feature (e.g. USA, UK, MEX, etc...)

Led by : Shreyas Shukla

Transforming Information

- We can use two approaches here:
  - Integer Encoding
  - One-hot Encoding (Dummy Variables)

Integer Encoding

- Directly convert categories into integers 1,2,3...N

Country
USA
MEX
CAN
USA

Country
USA
MEX
CAN
USA

Country
1
2
3
1

Possible issue is implied ordering and relationship (ordinal variable)

Country
USA
MEX
CAN
USA

→

Country
1
2
3
1

Here we see the implication MEX is twice the value of USA

Country
USA
MEX
CAN
USA

Country
1
2
3
1

PIERIAN DATA

Here we see the implication CAN is three times the value of USA

Country
USA
MEX
CAN
USA

Country
1
2
3
1

This may or may not make sense depending on the feature and domain

Country
USA
MEX
CAN
USA

Country
1
2
3
1

Spice Level
Mild
Hot
Fire
Mild

Spice Level
1
2
3
1

Integer Encoding

Always carefully consider the implication of integer encoding

Spice Level
Mild
Hot
Fire
Mild

Spice Level
1
2
3
1

Integer Encoding

- Pros:
  - Very easy to do and understand.
  - Does not increase number of features.
- Cons:
  - Implies ordered relationship between categories.

One Hot Encoding (Dummy Variables)

Convert each category into individual features that are either 0 or 1

Country
USA
MEX
CAN
USA

Country
USA
MEX
CAN
USA

USA	MEX	CAN
1	0	0
0	1	0
0	0	1
1	0	0

No ordered relationship is implied between categories.

Country
USA
MEX
CAN
USA

USA	MEX	CAN
1	0	0
0	1	0
0	0	1
1	0	0

However we greatly expanded our feature set, many more columns.

Country
USA
MEX
CAN
USA

USA	MEX	CAN
1	0	0
0	1	0
0	0	1
1	0	0

We can try to reduce this feature column expansion by creating higher level categories.

For example, regions or continents instead of countries.

Using pandas .map() or .apply() can achieve this.

May require a lot of tuning and domain experience to choose reasonable higher level categories or mappings.

Also must be aware of the “dummy variable trap”, mathematically known as multi-collinearity.

Converting to dummy variables can cause features to be duplicated.

Let’s consider the simplest possible example...

Consider a binary category (only two options):

Vertical Direction
UP
DOWN
UP
DOWN

Vertical Direction
UP
DOWN
UP
DOWN

UP	DOWN
1	0
0	1
1	0
0	1

The new columns are duplicate information with inverted encoding.

Vertical Direction
UP
DOWN
UP
DOWN

UP	DOWN
1	0
0	1
1	0
0	1

Easily fixed by simply dropping last column.

Vertical Direction
UP
DOWN
UP
DOWN

UP
1
0
1
0

This can be extended to more than 2 categories:

Country
USA
MEX
CAN
USA

USA	MEX
1	0
0	1
0	0
1	0

One Hot Encoding (Dummy Variables)

- Pros:
  - No ordering implied.
- Cons:
  - Potential to create many more feature columns and coefficients.
  - Dummy variable trap consideration.
  - Not easy to add new categories.

- Keep in mind feature engineering in general will always be data and domain dependent.
- There is no one size fits all solution!

Let’s get started!

Mastering Machine Learning with Python

(27th Aug 2024 - 18th Oct 2024)

iHUB DivyaSampark, IIT Roorkee

and

Support Vector Machines

Led 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

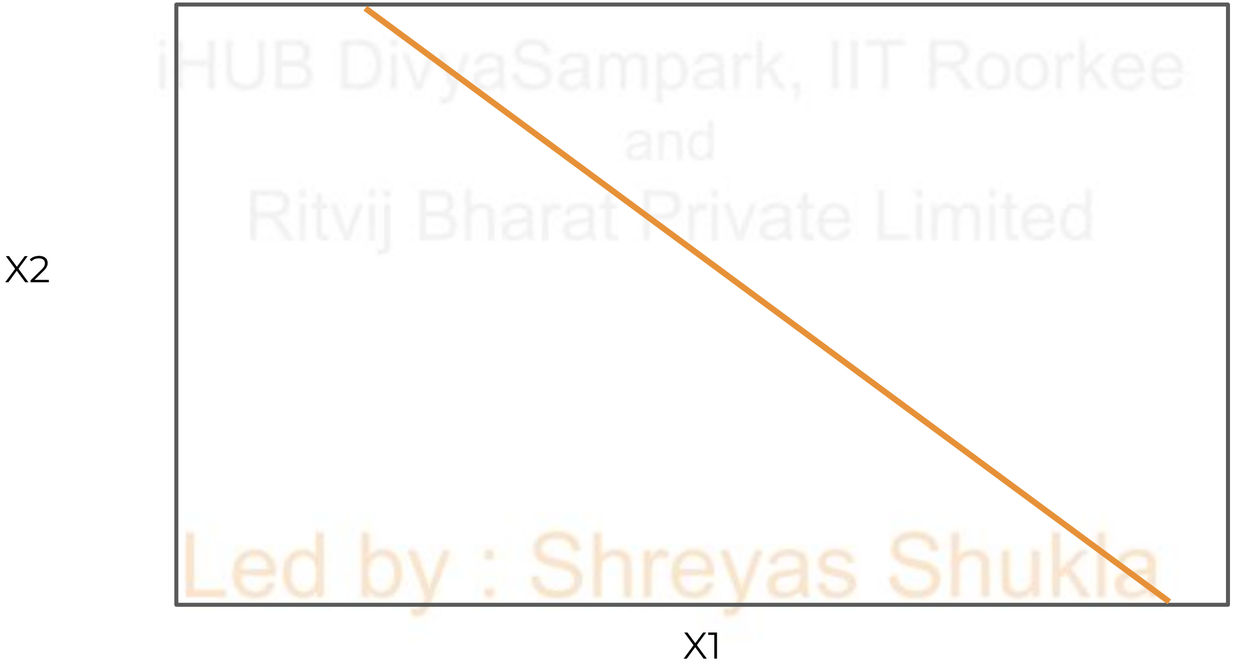
1-D Hyperplane

2-D Hyperplane

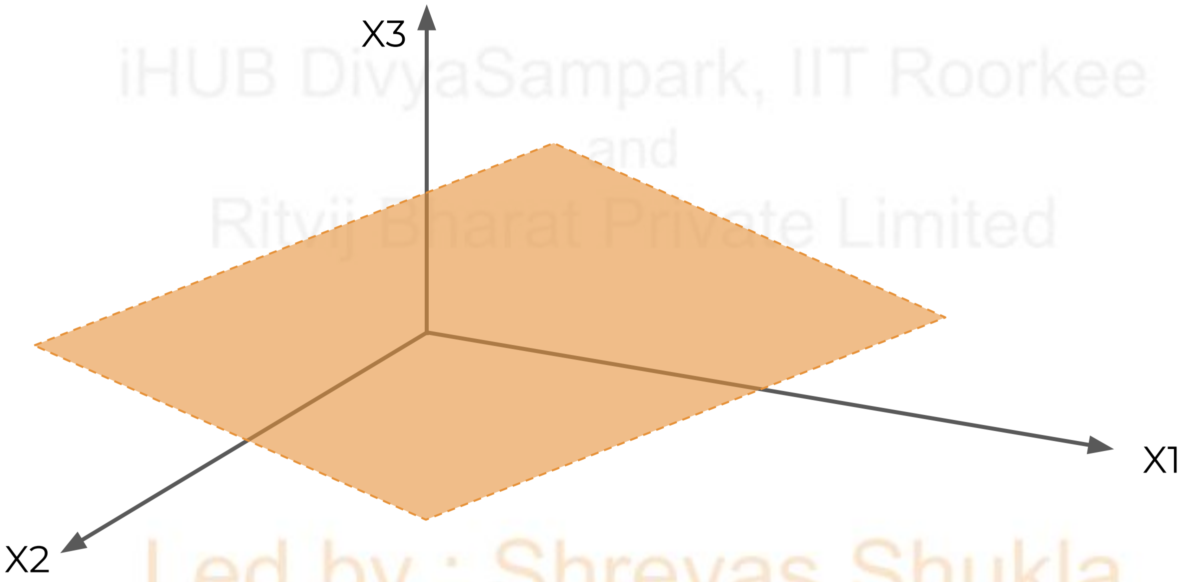
X2

X1

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:

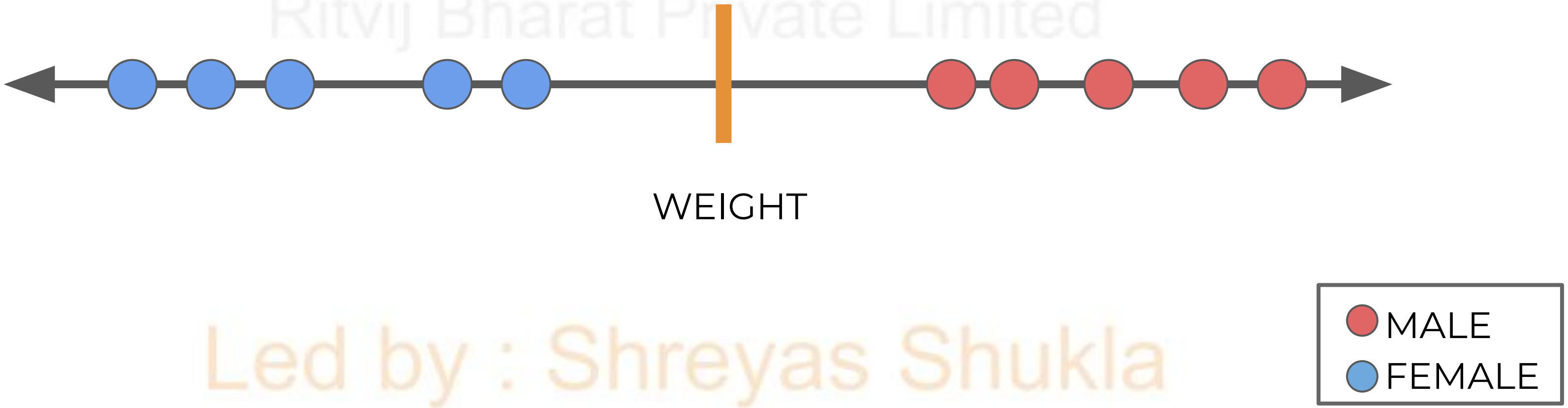
- A weight feature for baby chicks
- Classified by Male or Female

What would this be visualized?

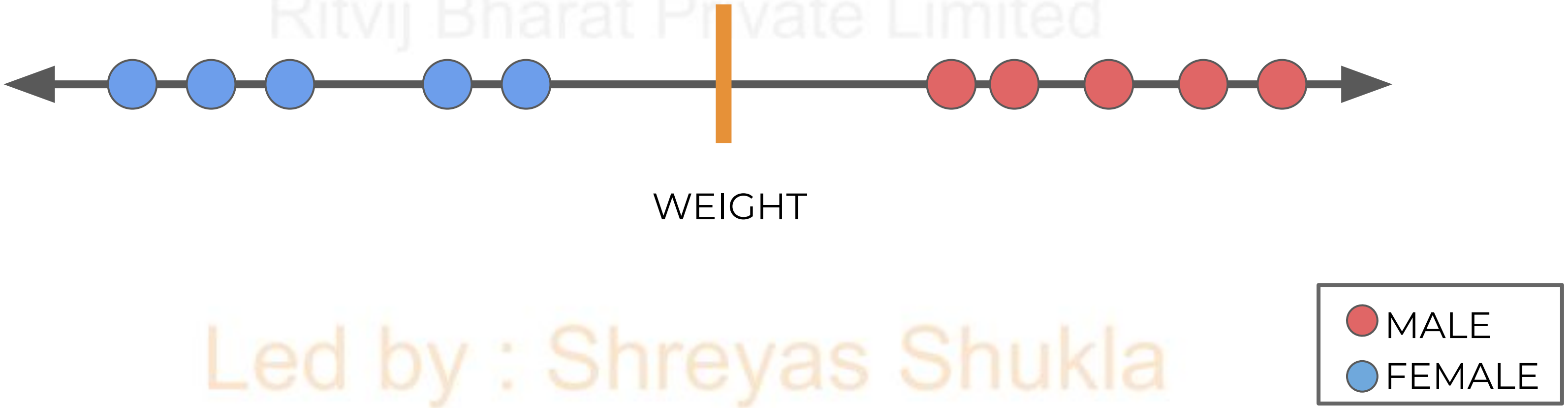
Place points along feature.

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

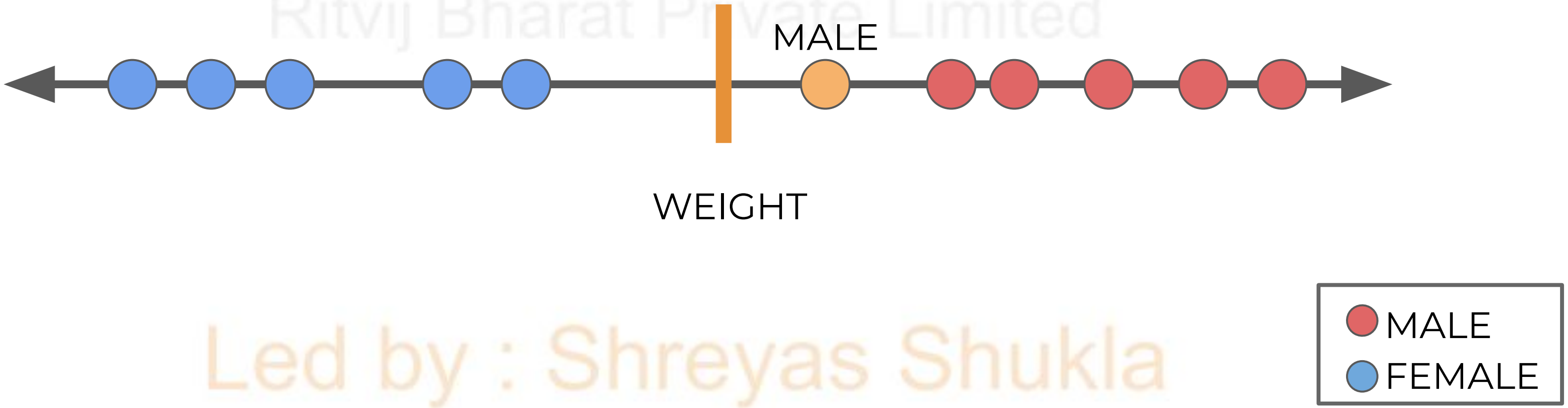
Idea behind SVM is to create a **hyperplane** that will separate the classes.



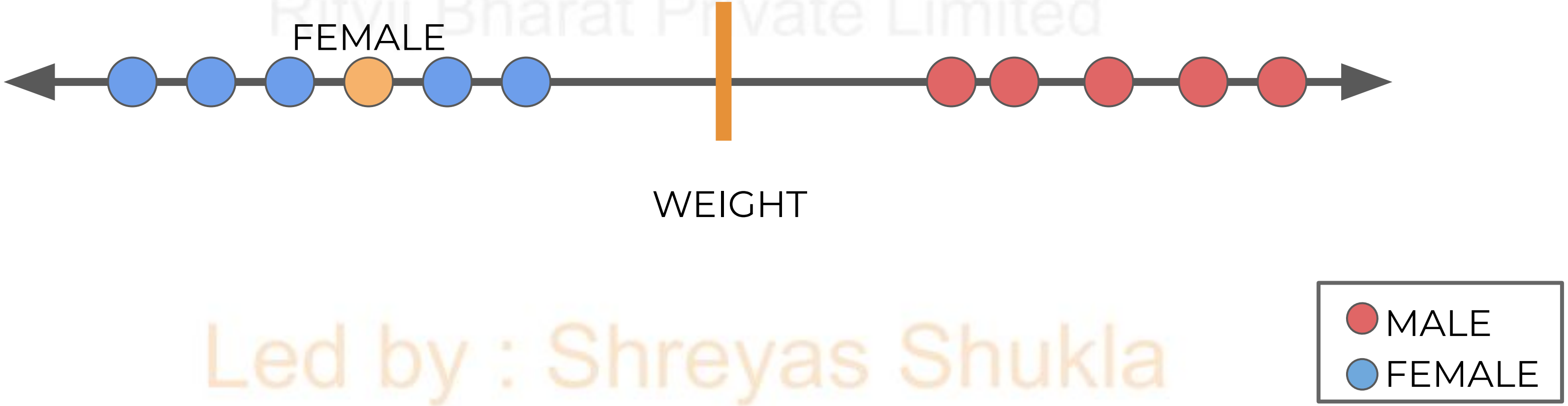
The classification of a new point is determined by the side of the hyperplane on which it falls.



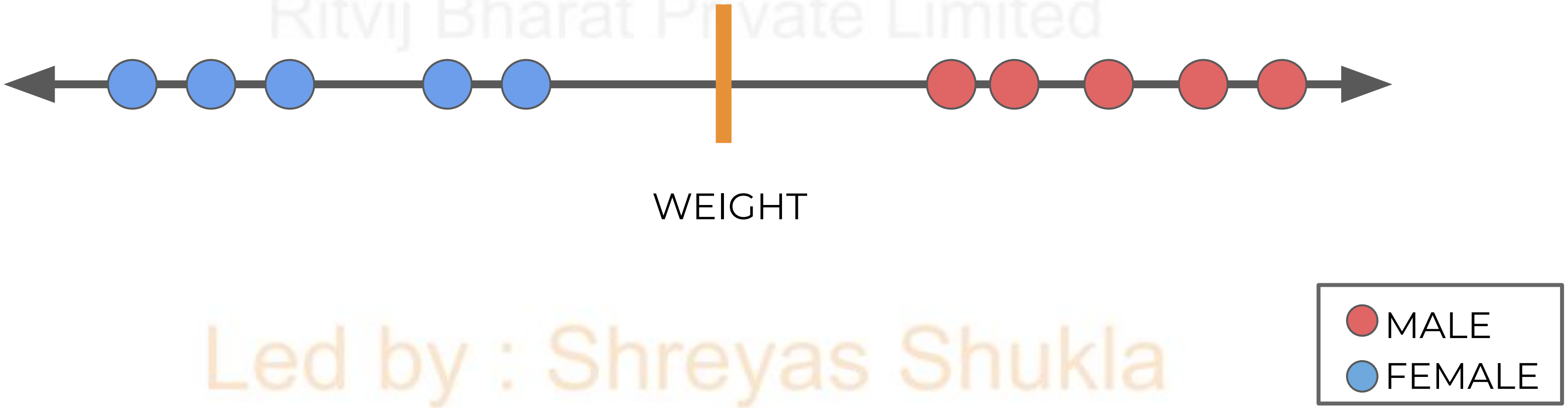
The classification of a new point is determined by the side of the hyperplane on which it falls.



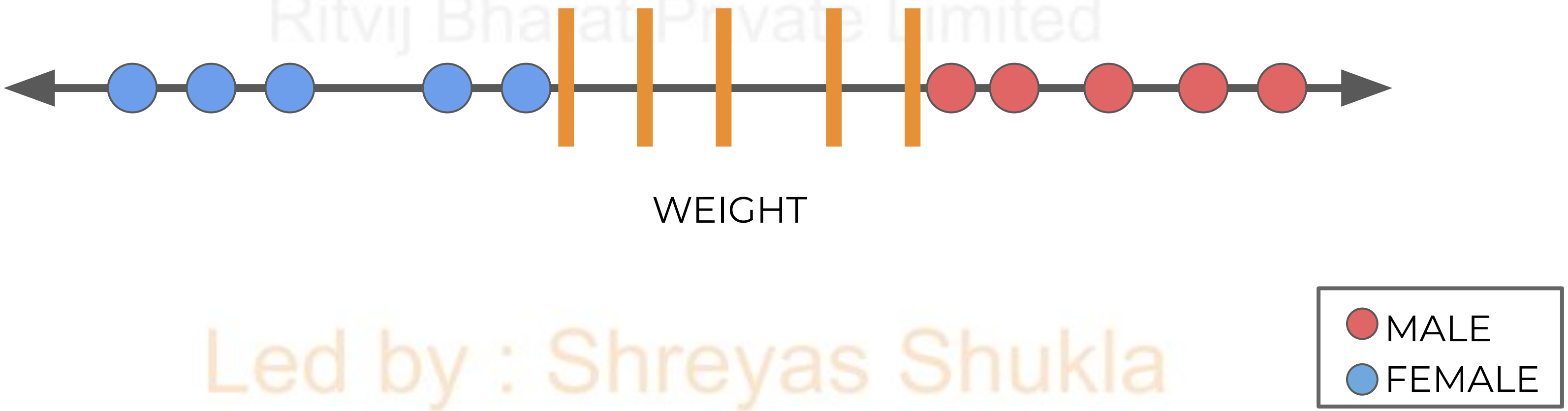
The classification of a new point is determined by the side of the hyperplane on which it falls.



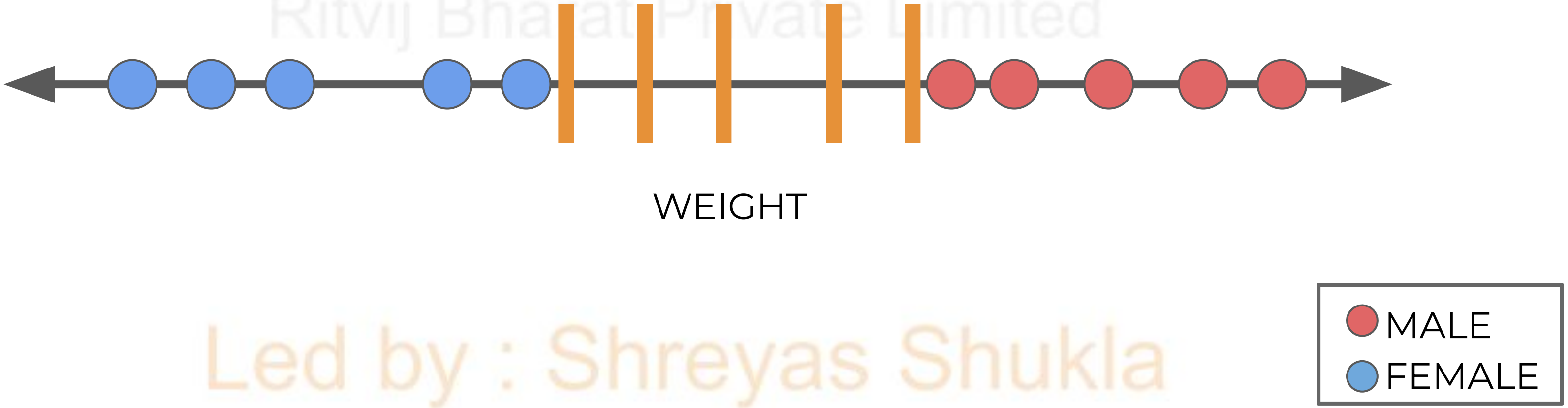
The classification of a new point is determined by the side of the hyperplane on which it falls.



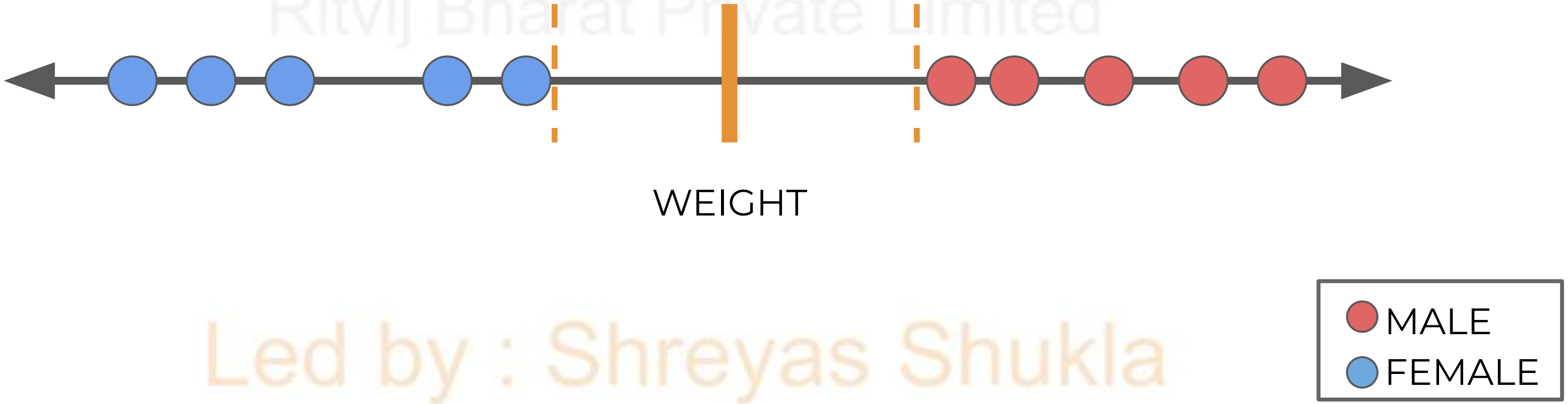
You'll notice that there are many options that perfectly separate out these classes



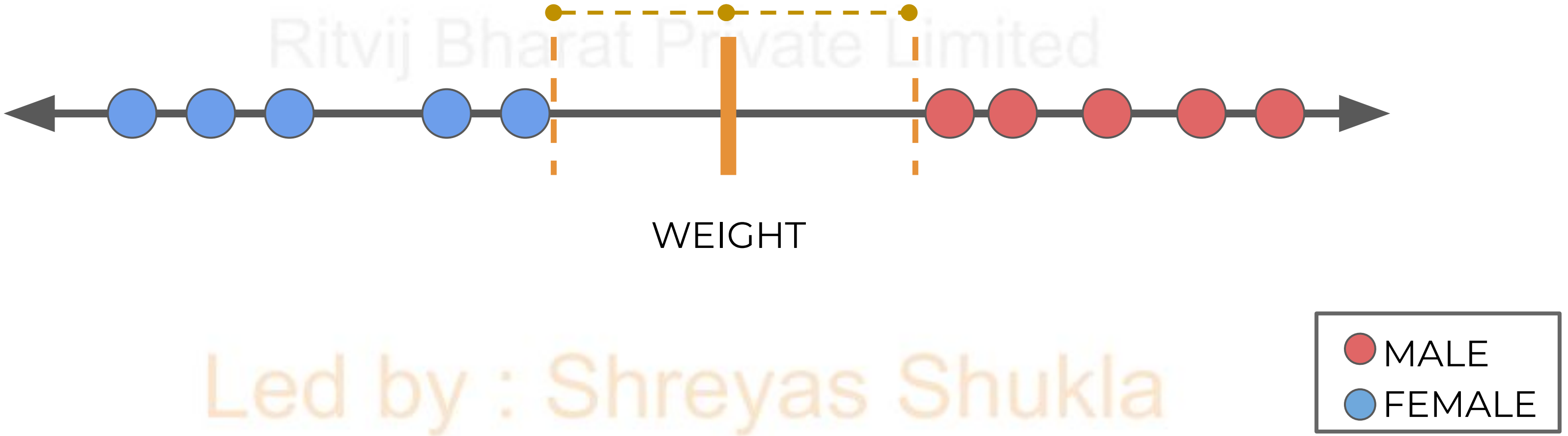
Which one is the “best” separator between the classes?



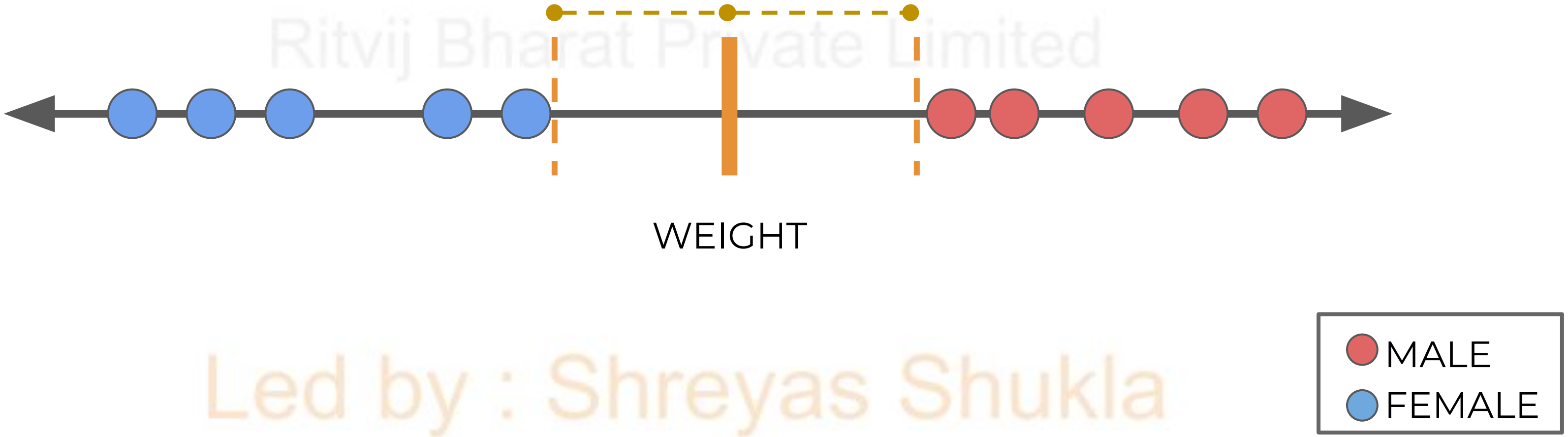
Use the separator that **maximizes** the **margins** between the classes.



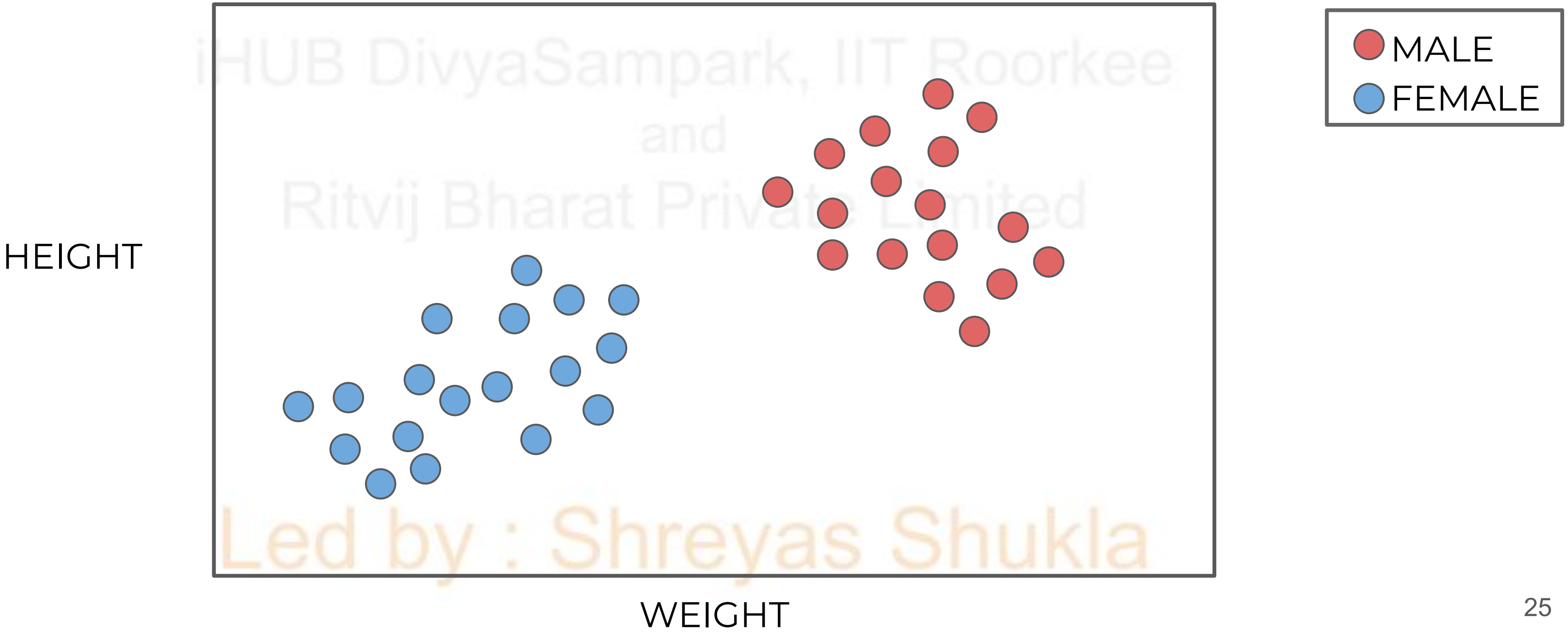
Maximal Margin Classifier.



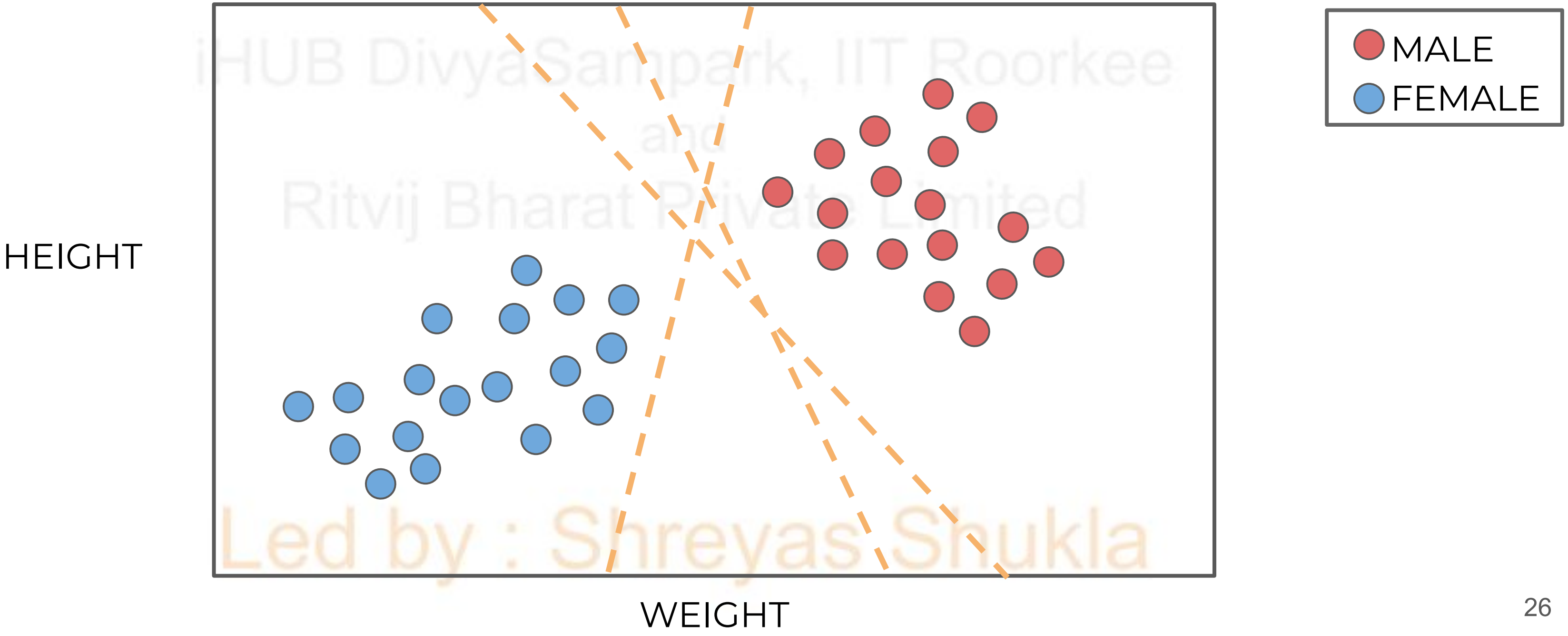
This very idea of maximum margins applies to N-dimensions.



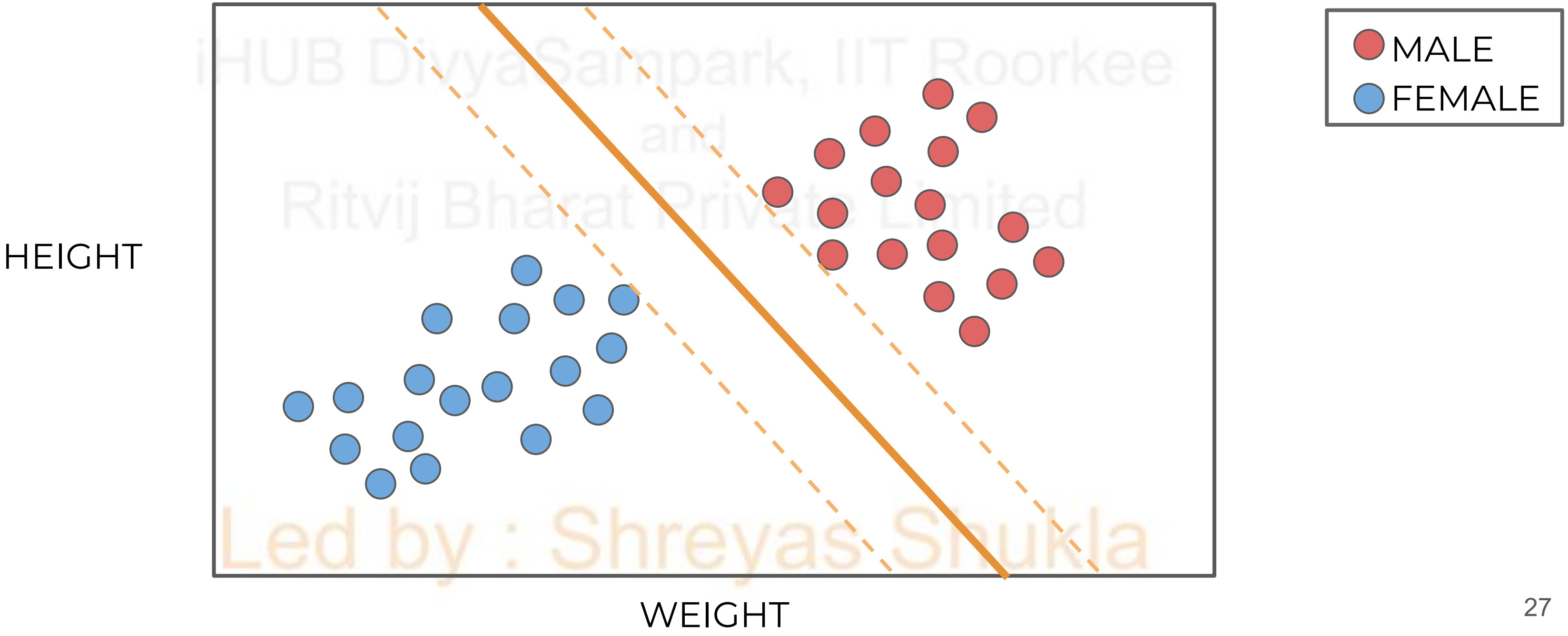
Imagine a 2 dimensional feature space:



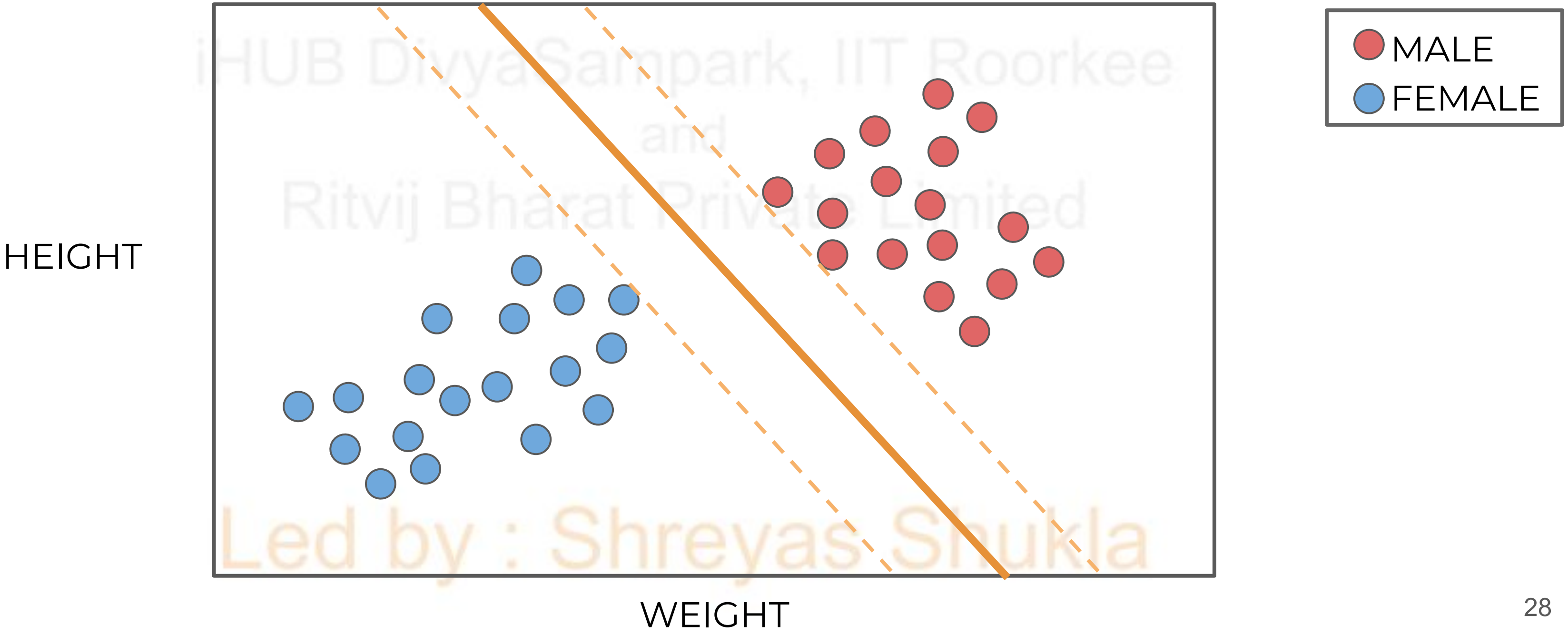
We could have Multiple possible hyperplanes:



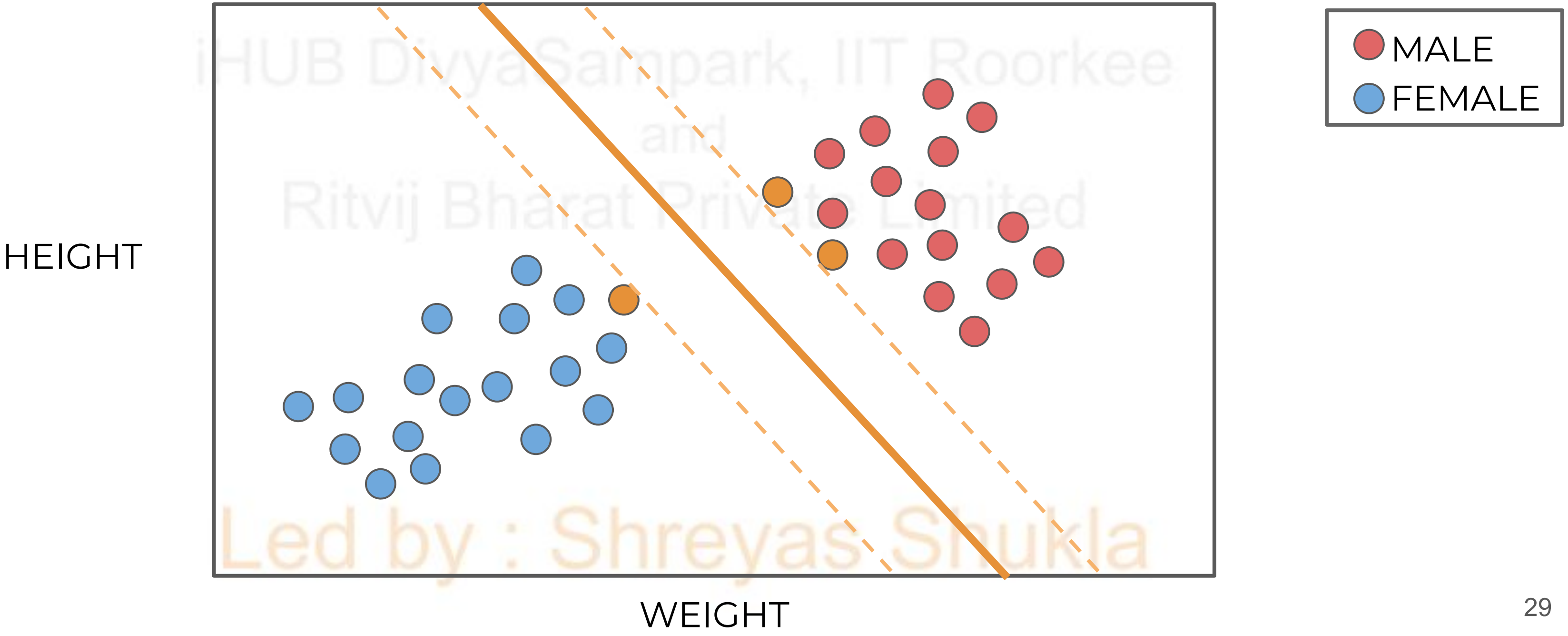
Choose to maximize margins:



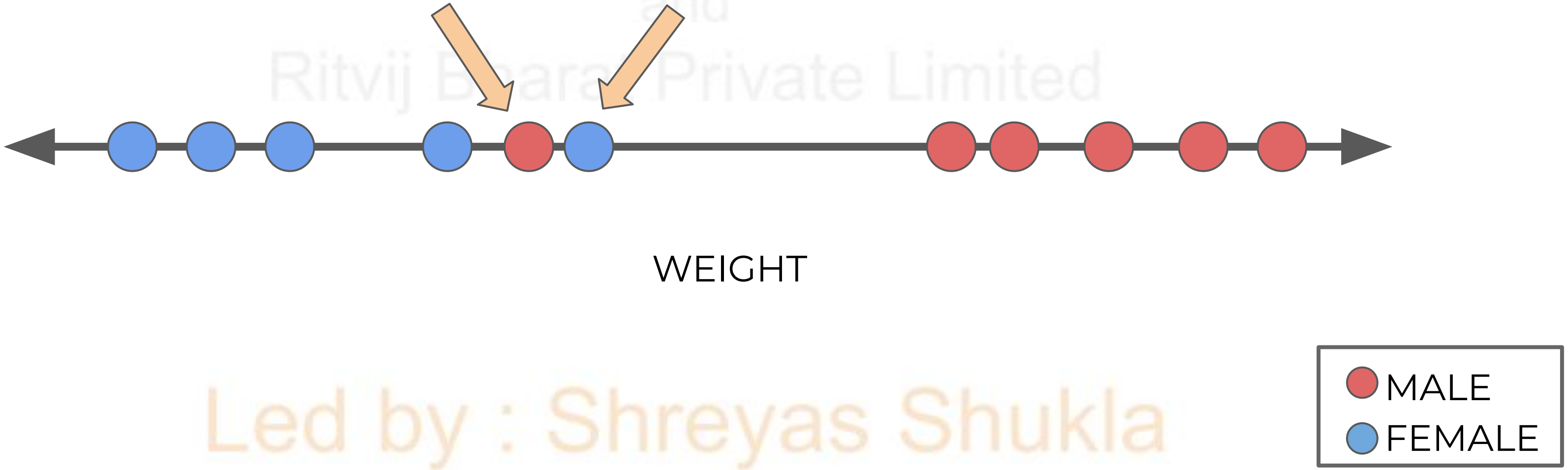
Note each data point is a 2D vector:



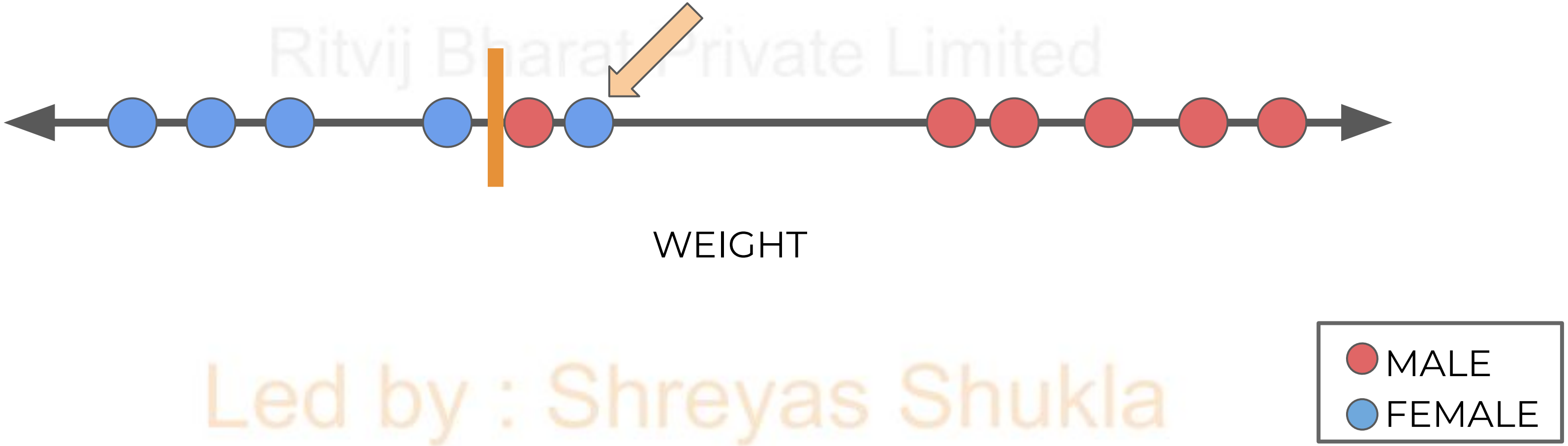
Data points at margin support separator:



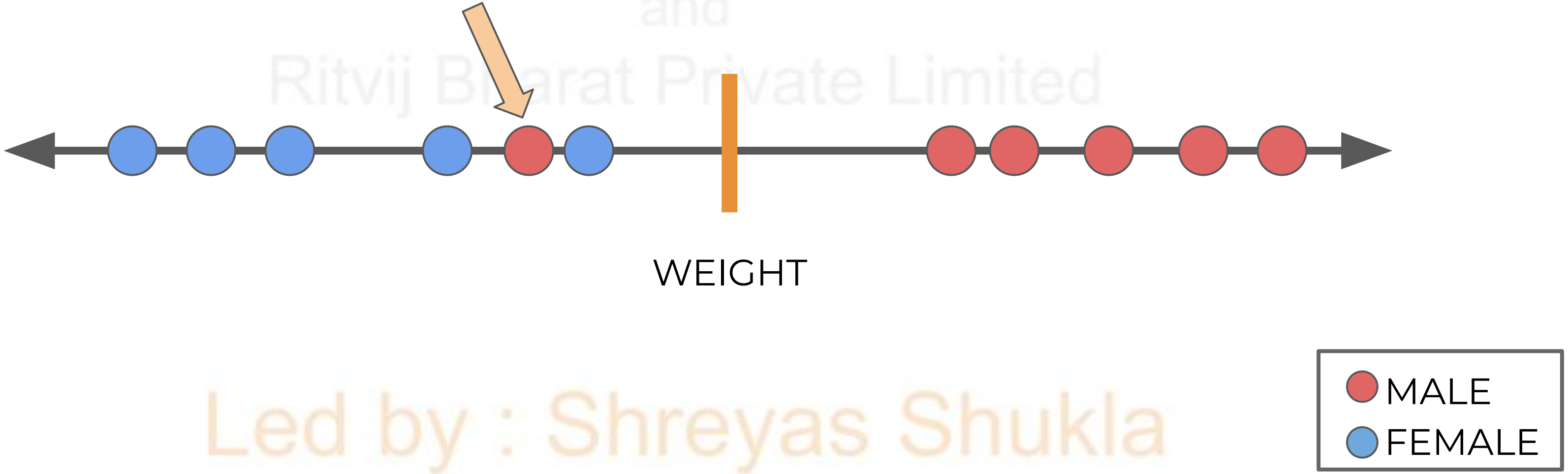
What happens if classes are not perfectly separable?



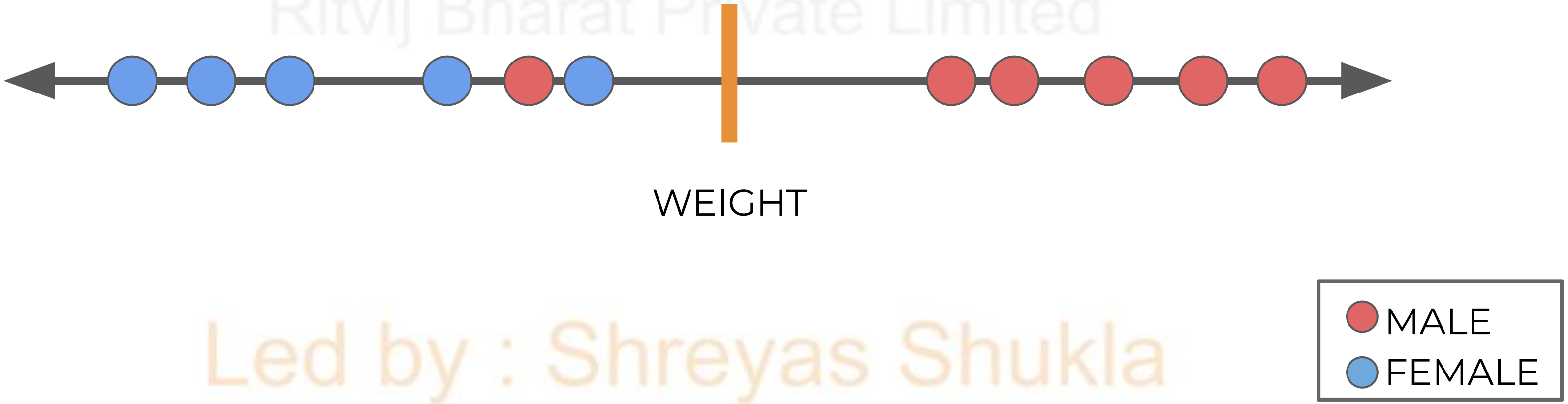
It is impossible to achieve perfect separation without accepting the possibility of misclassifications.



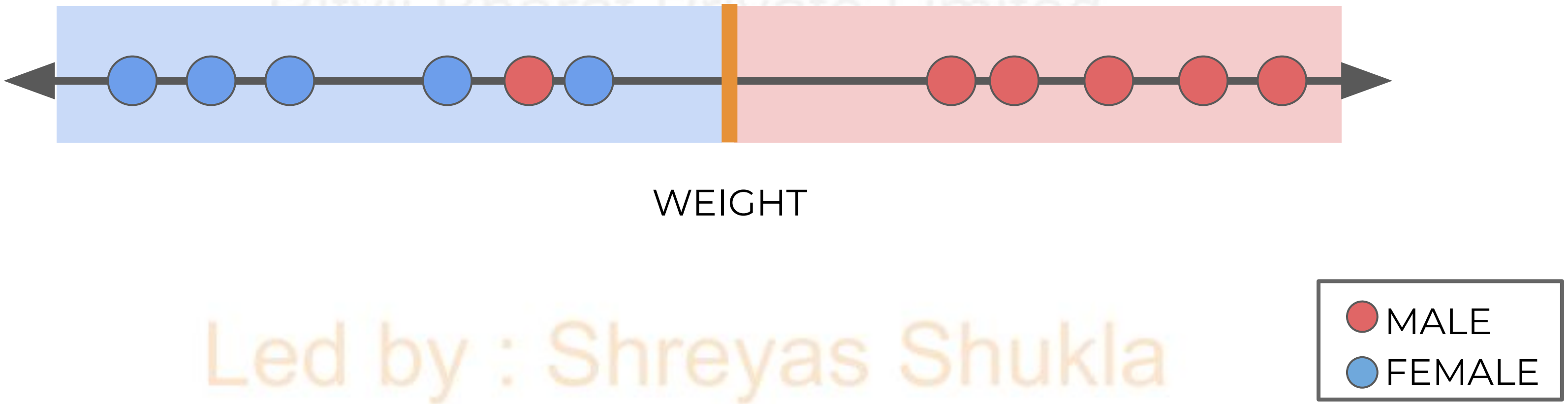
It is impossible to achieve perfect separation without accepting the possibility of misclassifications.



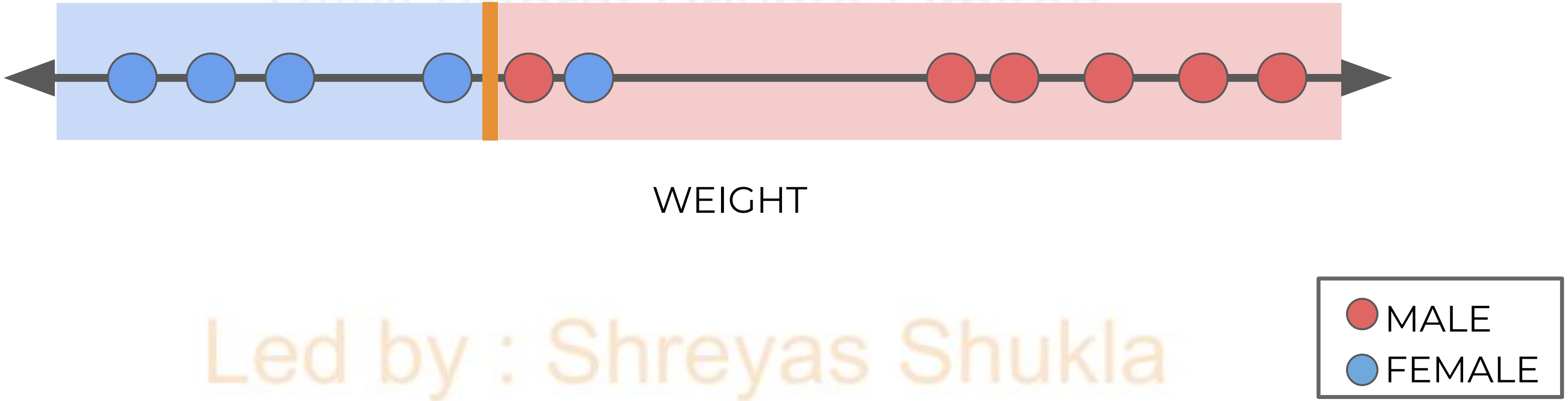
We encounter a bias-variance trade-off depending where we place this separator:



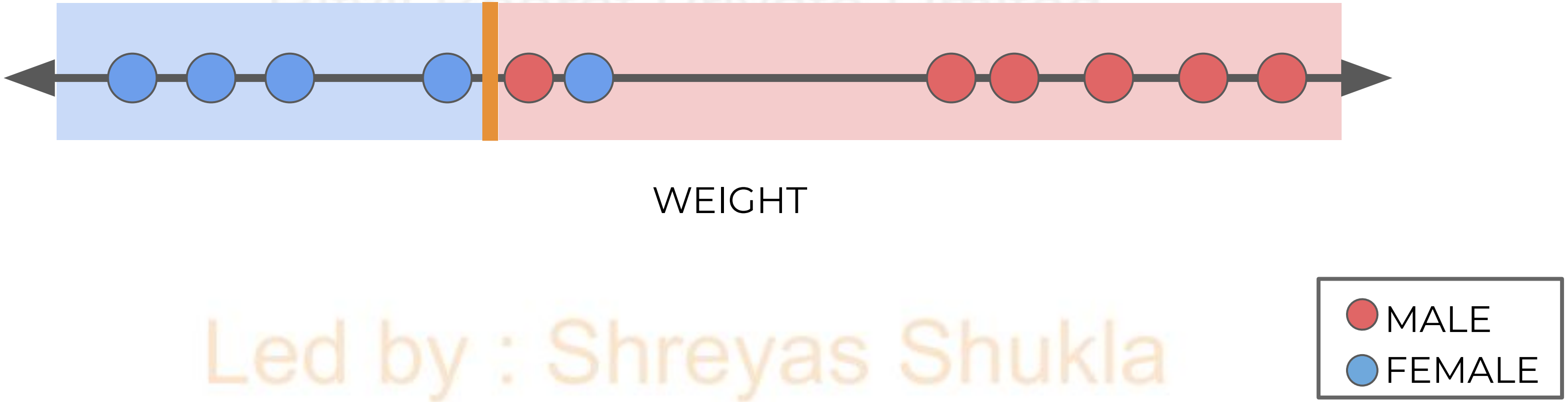
For one feature this classifier creates range for male and female:



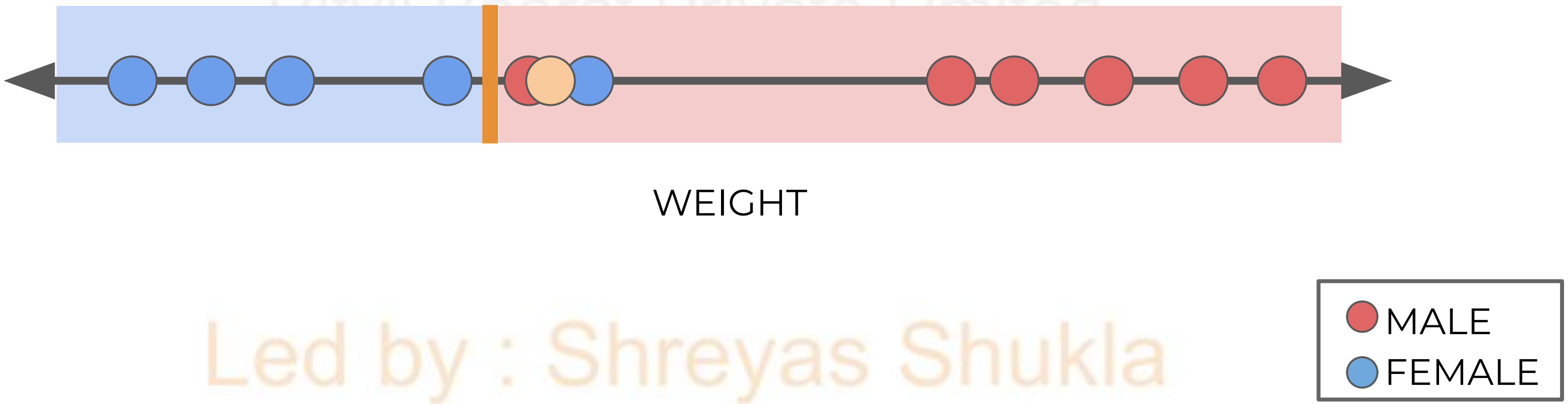
This fit only misclassified one female training point as male:

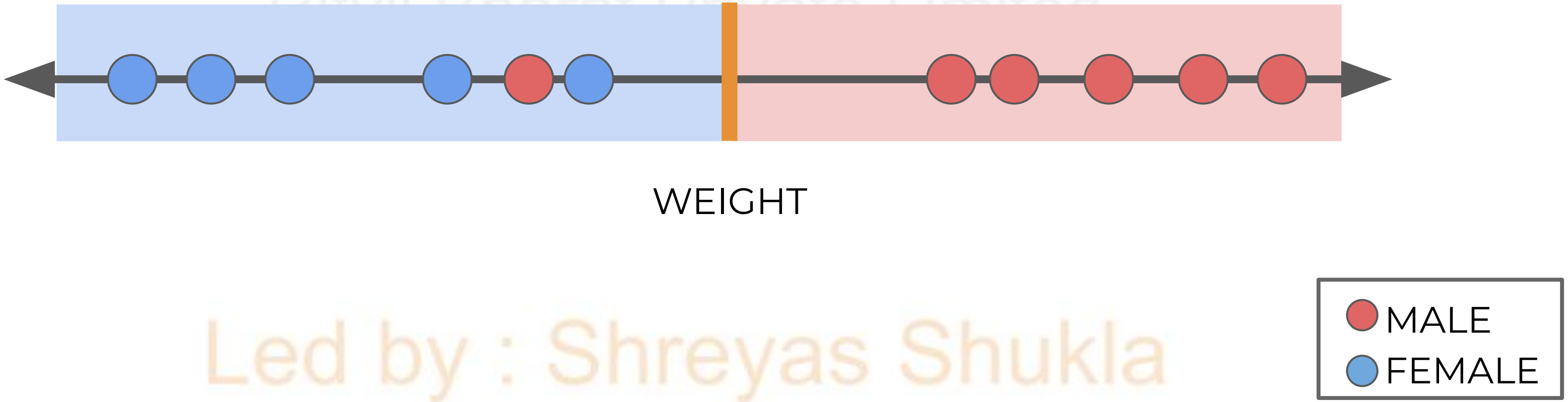


This is a high variance fit to training data, picking too much noise from Female:

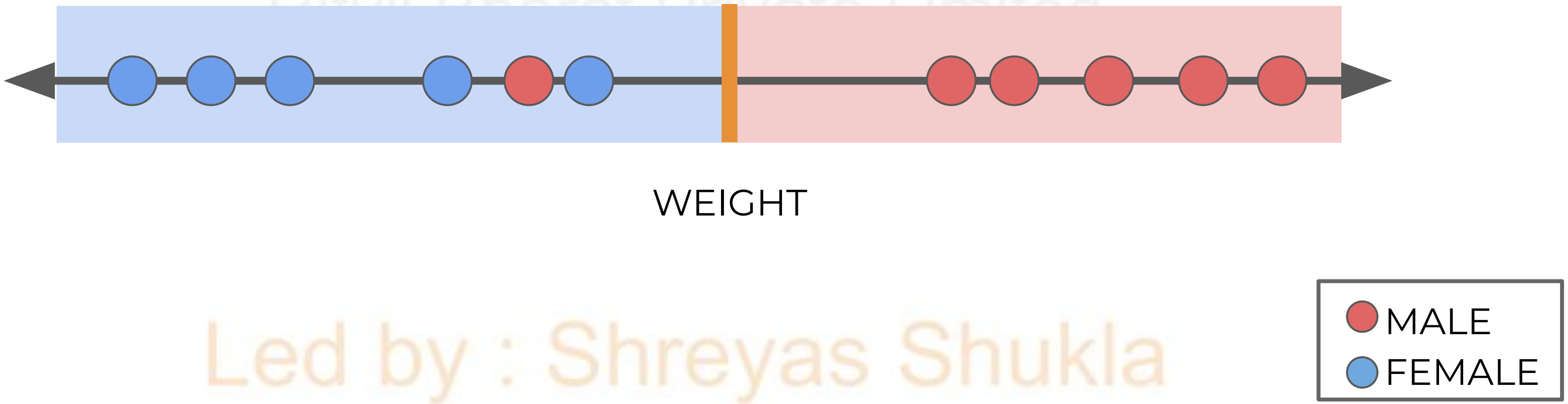


A new test point close to existing female weights could get classified as male:

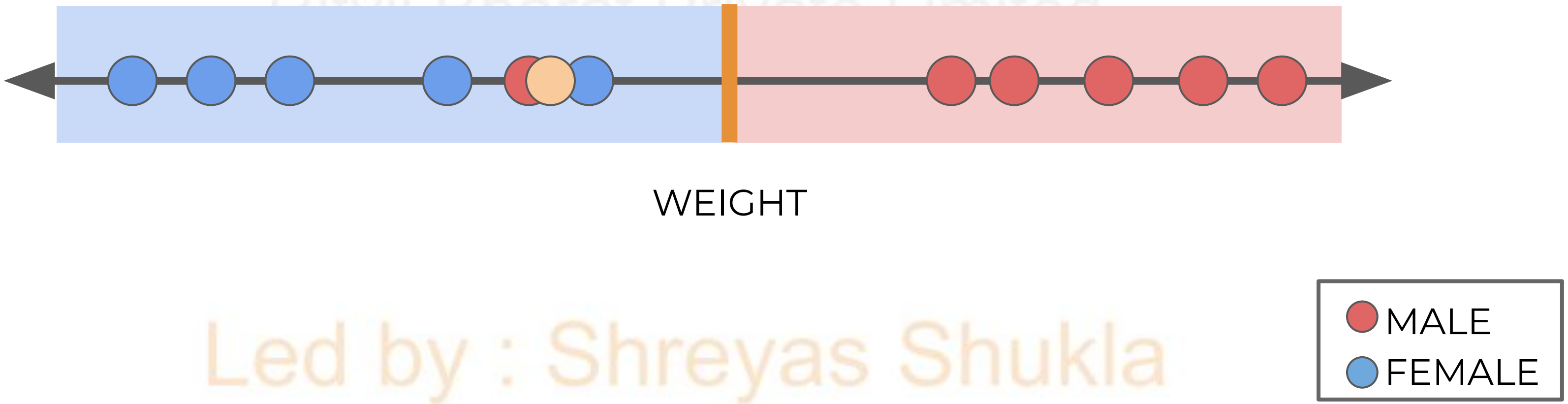



WEIGHT
MALE
FEMALE

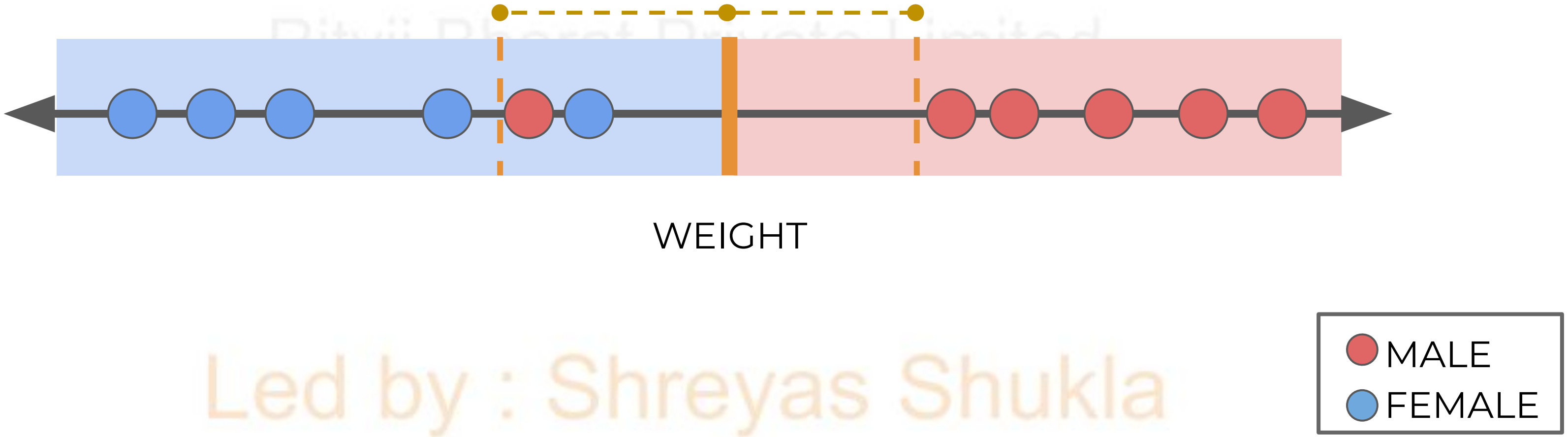
We allow more bias to achieve better long term results on future data:



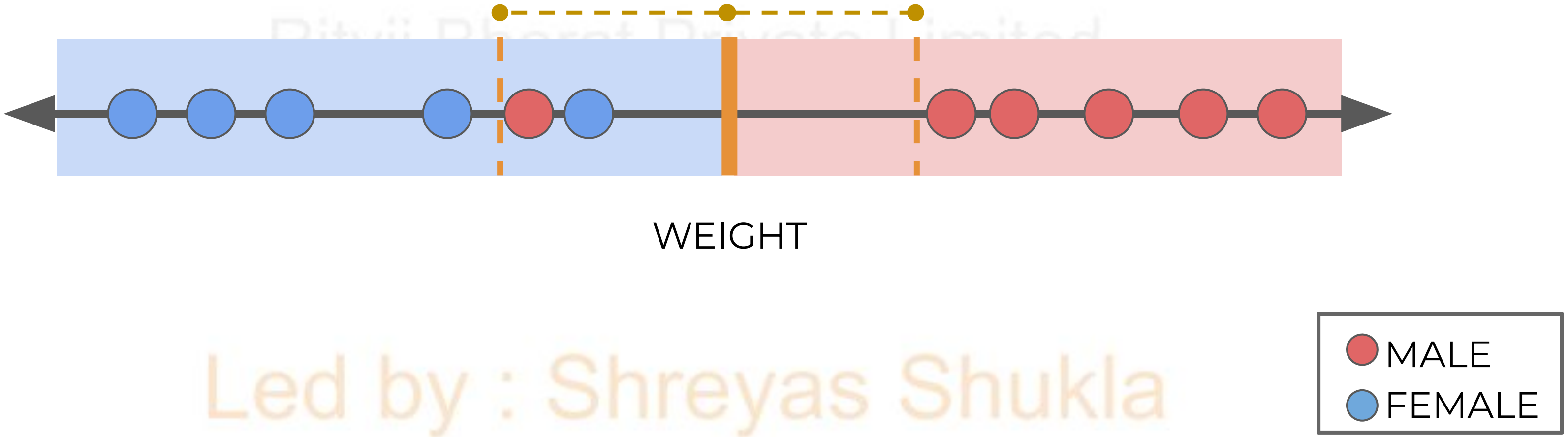
We allow more bias to achieve better long term results on future data:



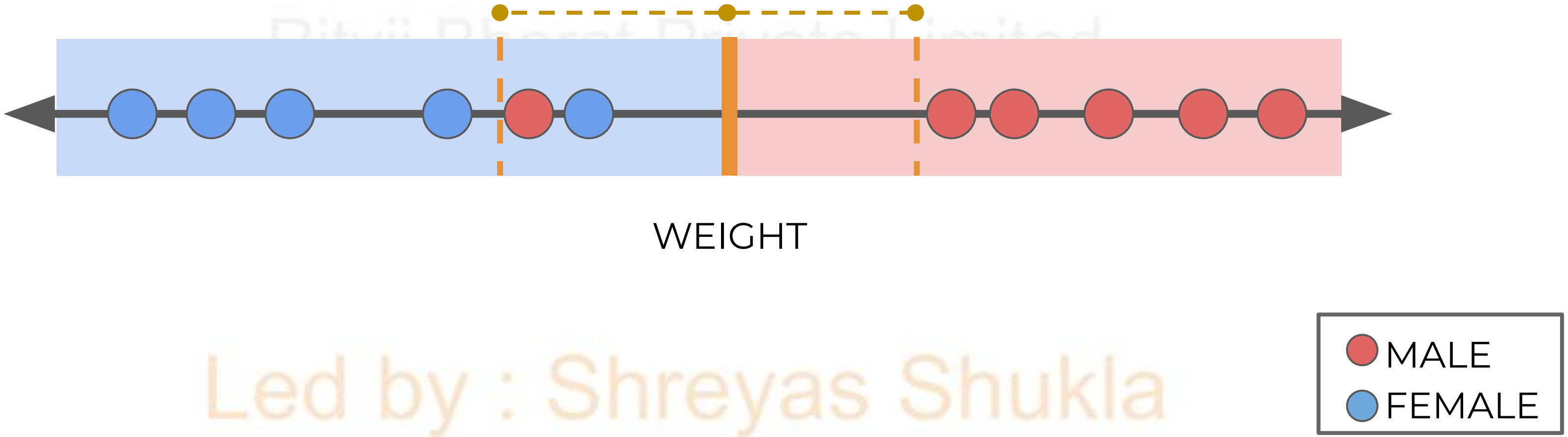
Soft margin: Distance between threshold and the observations



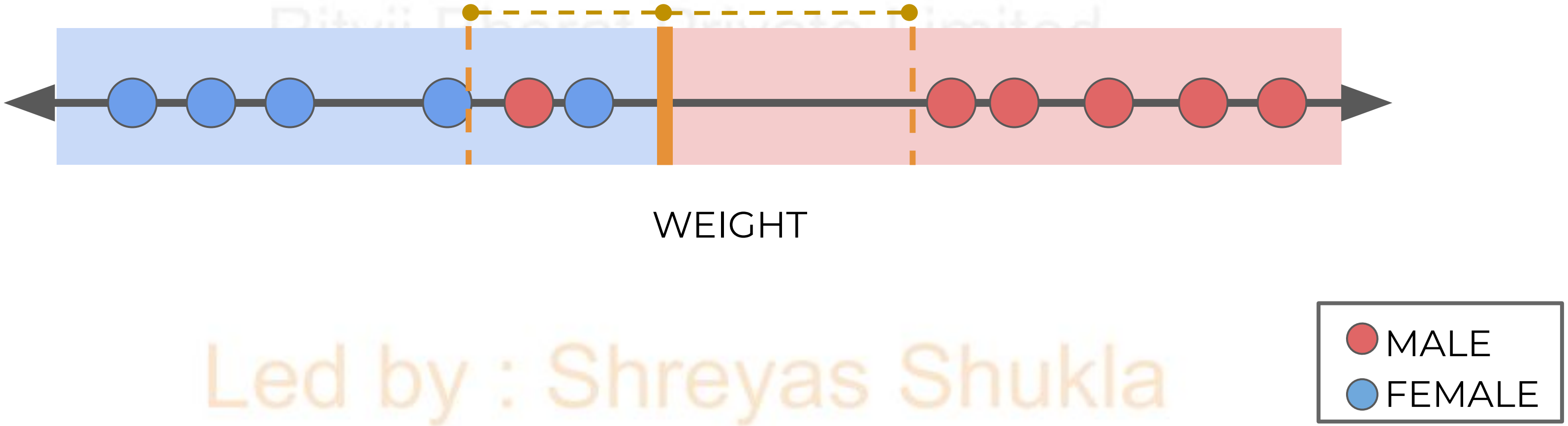
Soft margin: Distance between threshold and the observations



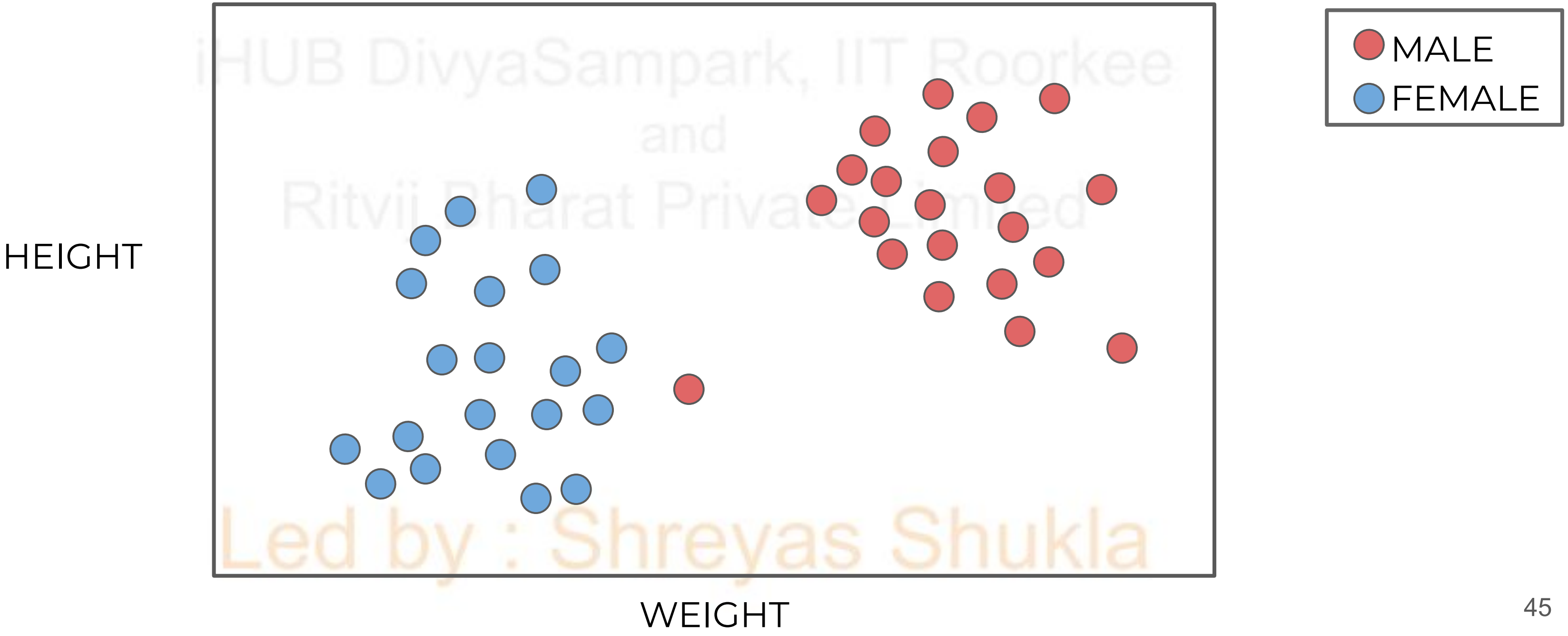
Many threshold splits possible if we allow for soft margins.



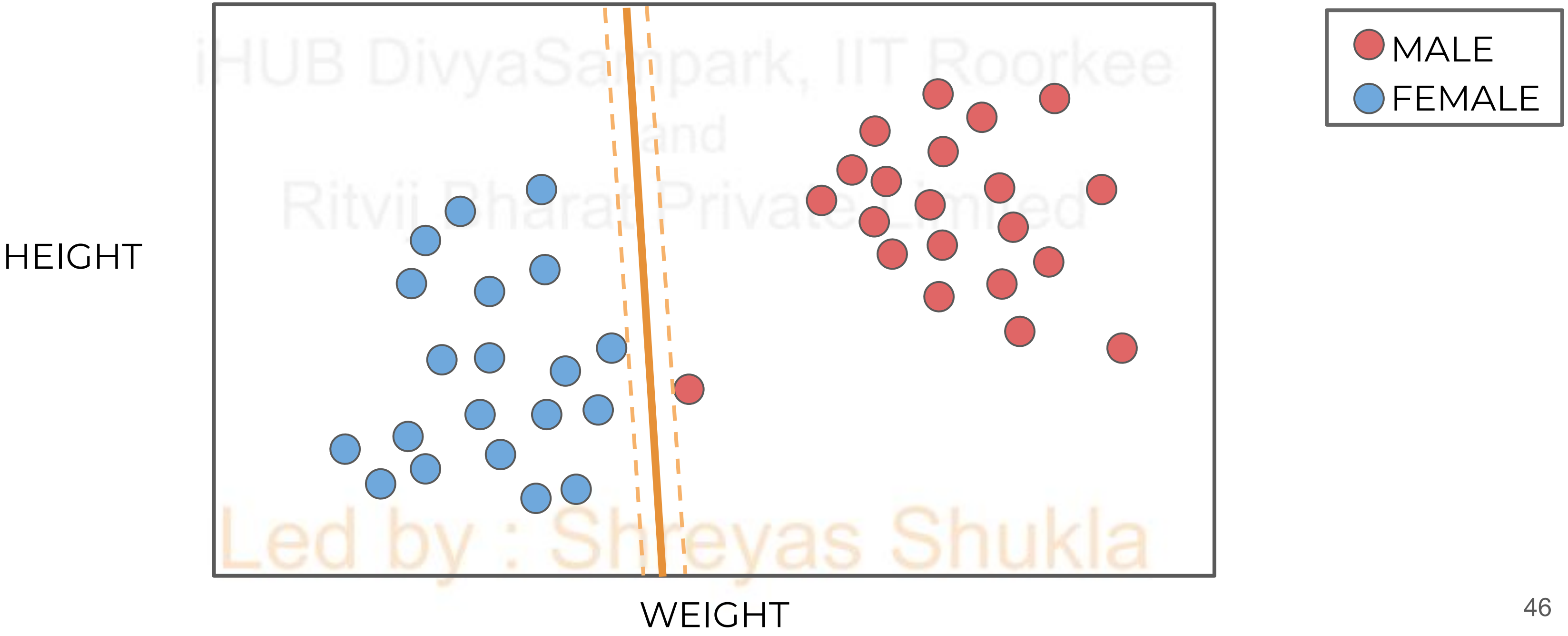
Use cross validation to determine the optimal size of the margins.



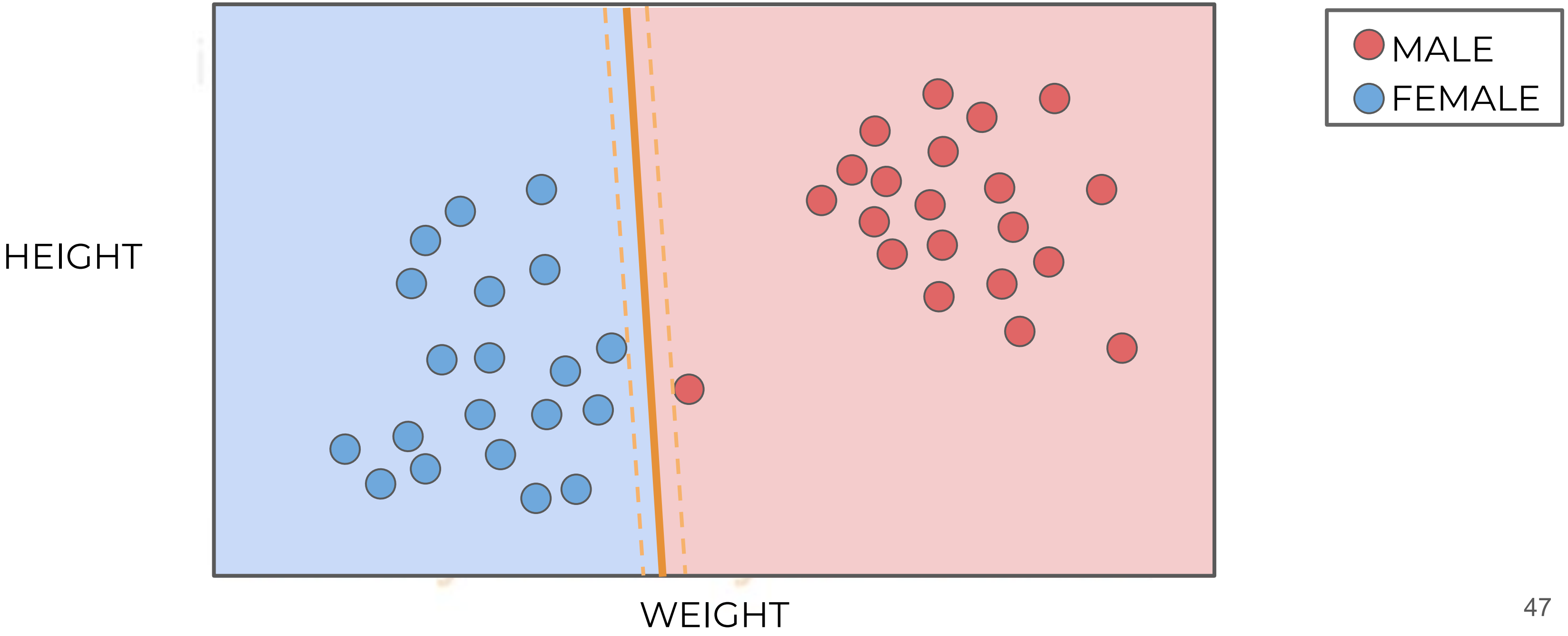
Here, dataset is technically perfectly separable



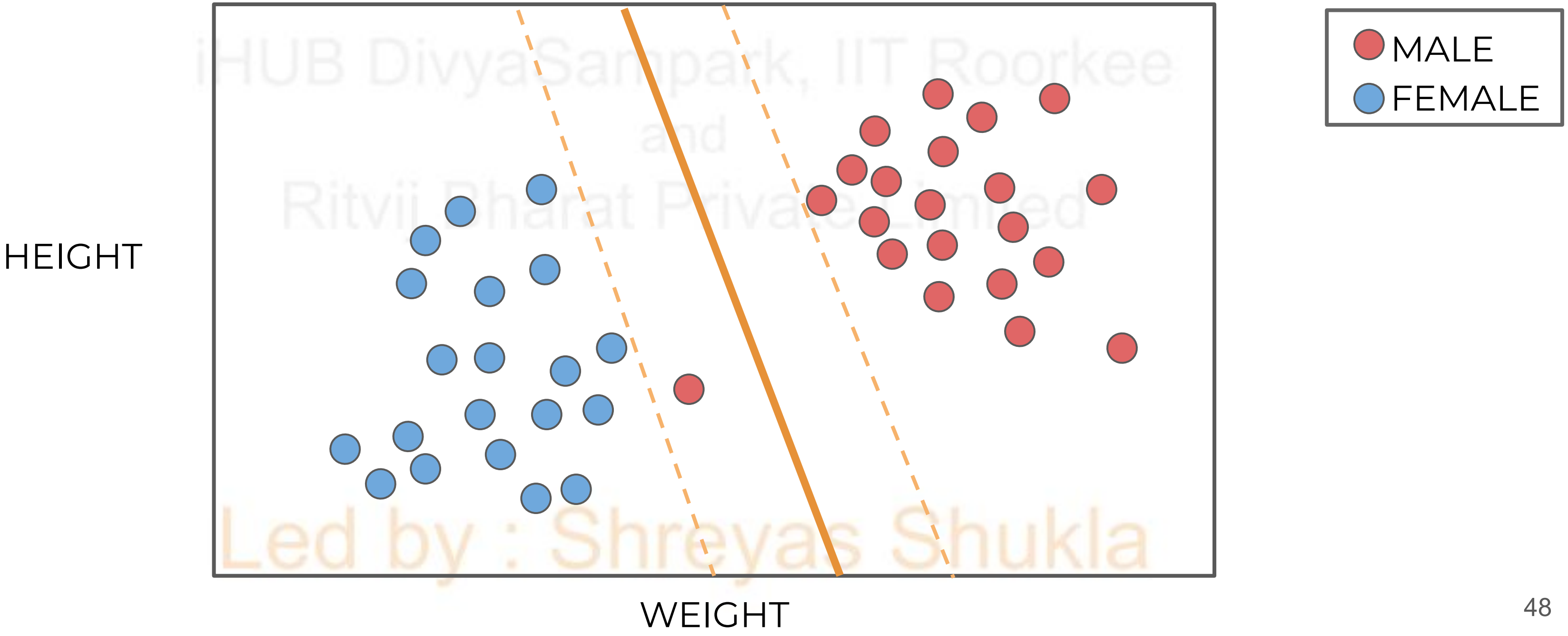
Maximal Margin Classifier



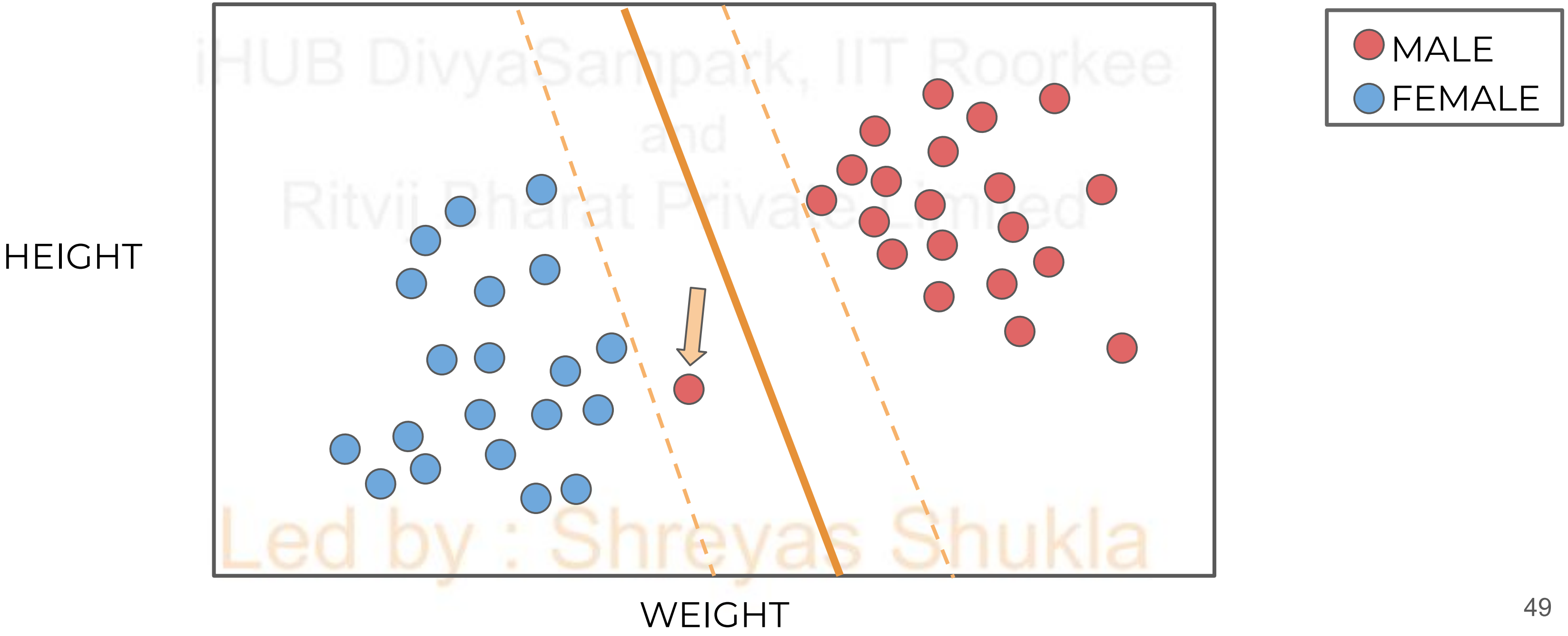
Maximal Margin Classifier



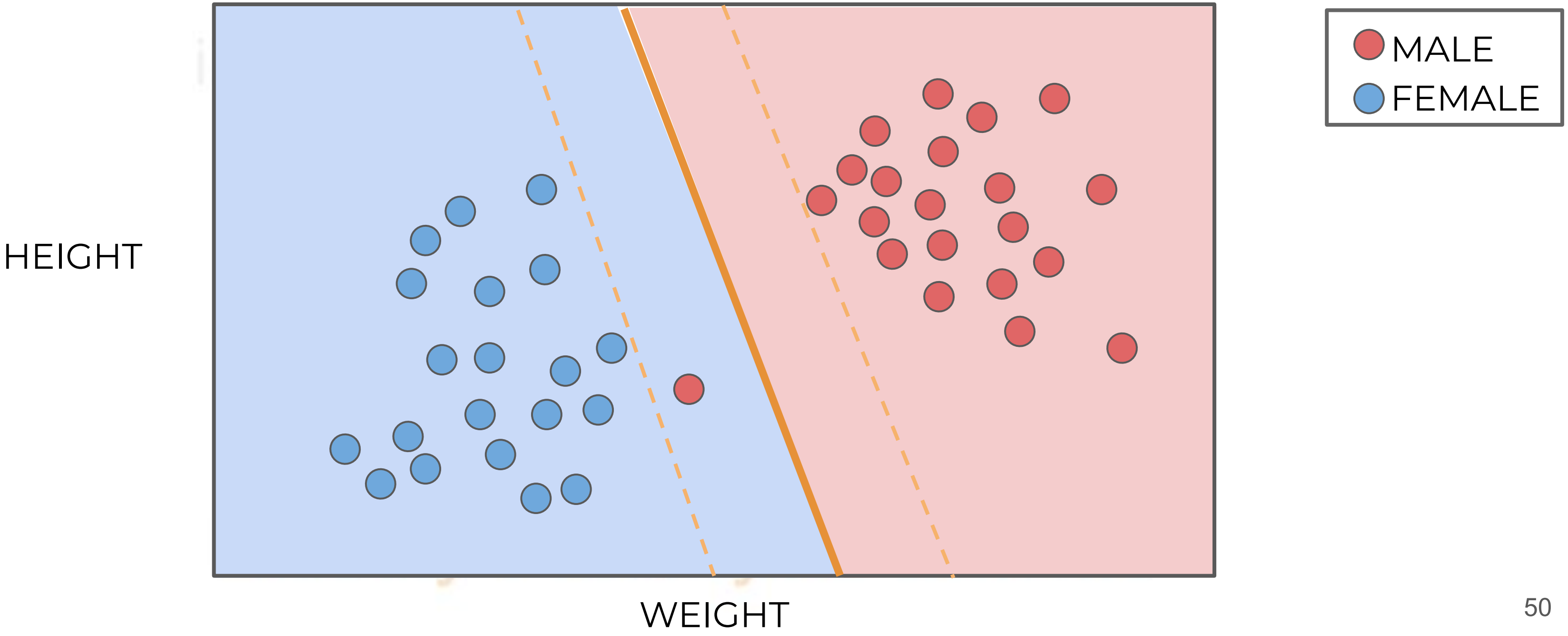
Support Vector Classifier (Soft Margins)



Support Vector Classifier (Soft Margins)

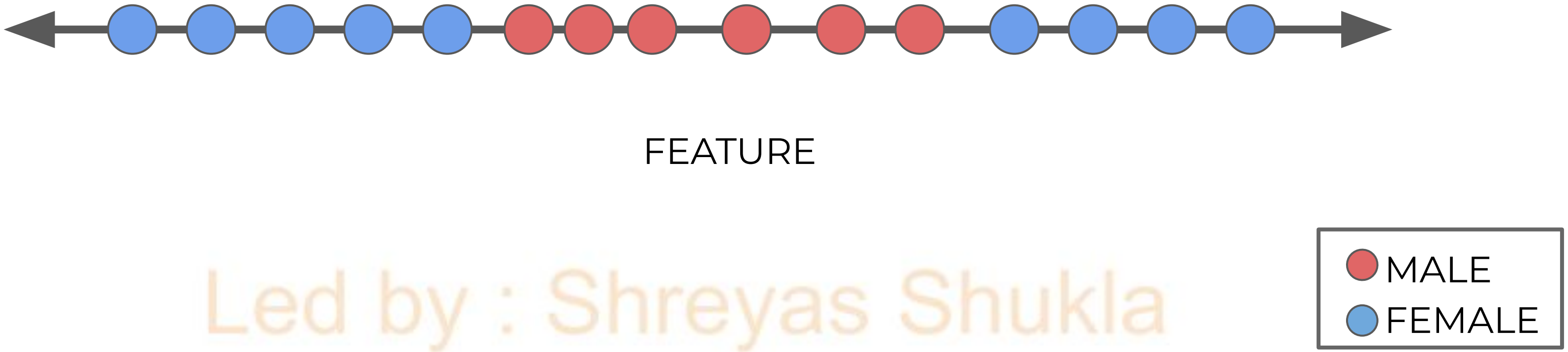


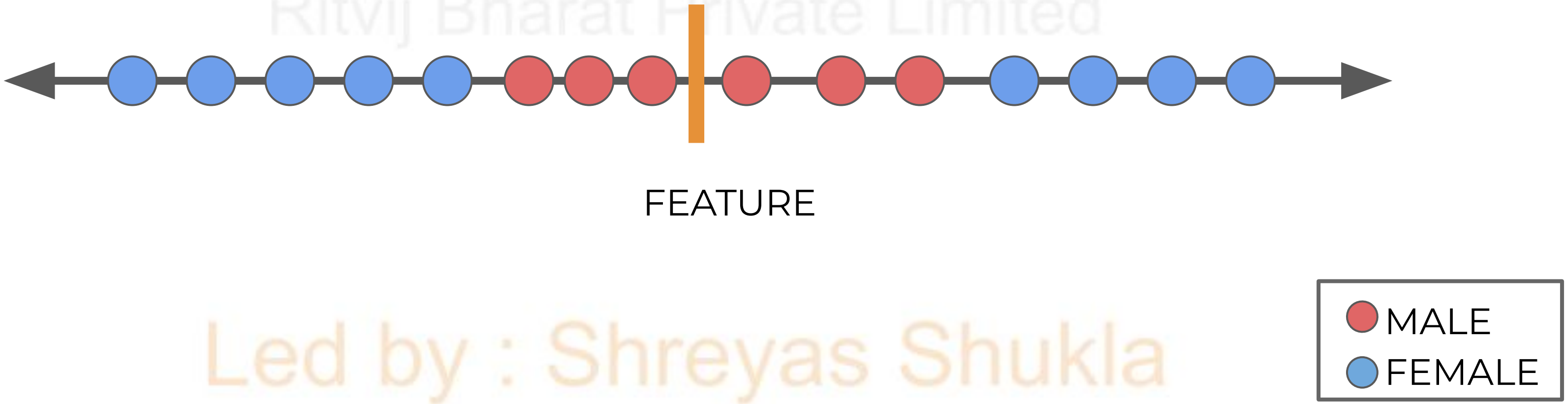
Support Vector Classifier (Soft Margins)

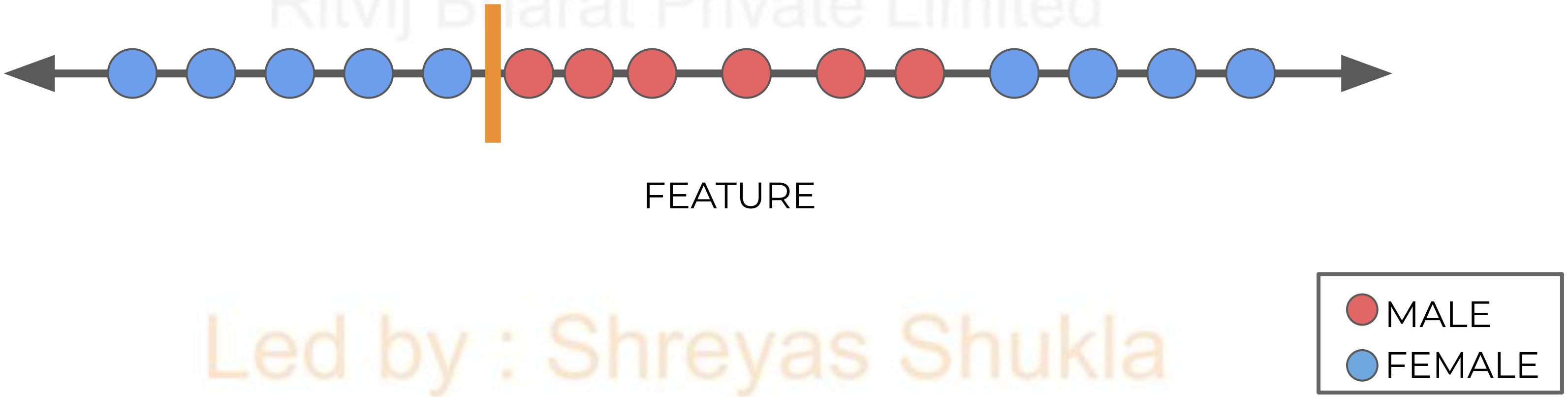


We've only visualized cases where the classes are easily separated by the hyperplane in the original feature space.
This leaves space for some misclassifications that will still result in reasonable results.
But what if a hyperplane performs poorly, even when allowing for misclassifications?

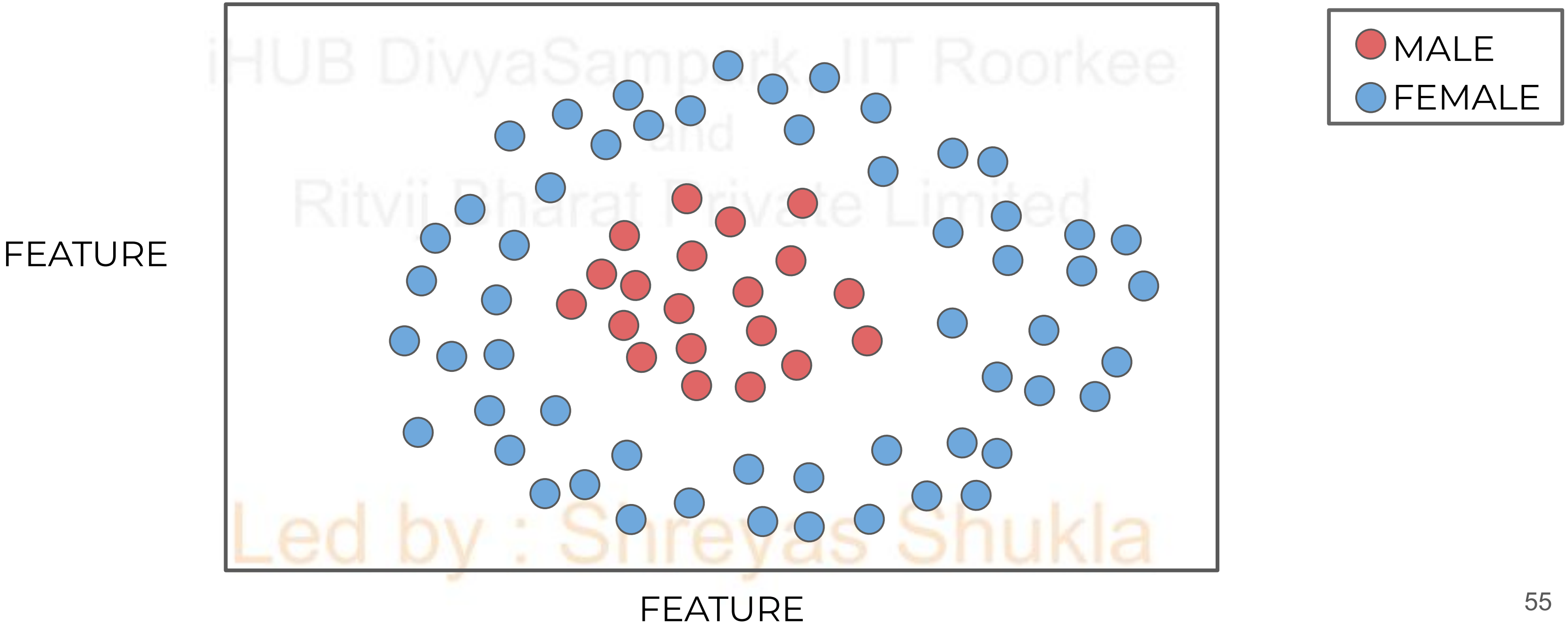
Notice a single hyperplane won't separate out the classes without many misclassifications!




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.

Support Vector Machines

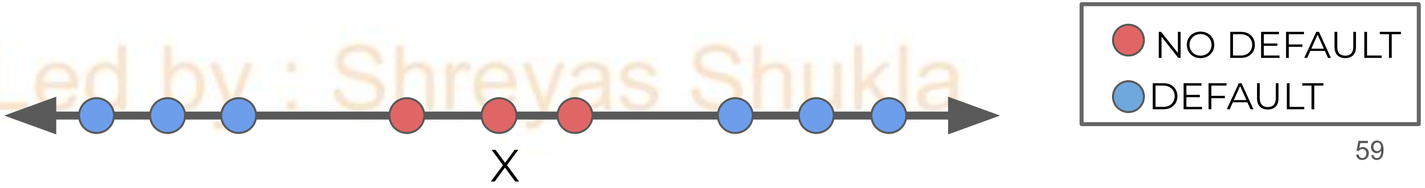
Theory and Intuition - Kernels

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

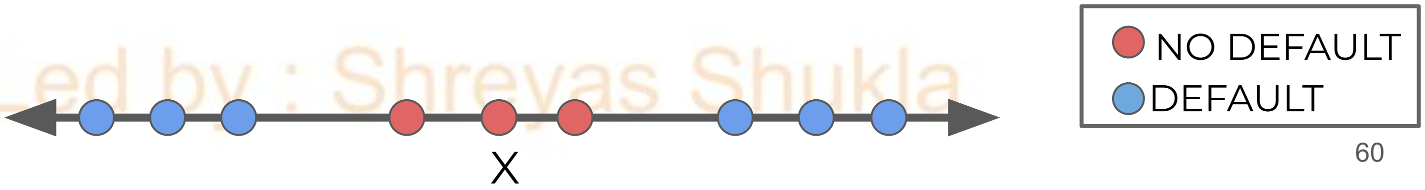
Variety of kernels can be used to “project” the features to a higher dimension.

Let’s see how this works

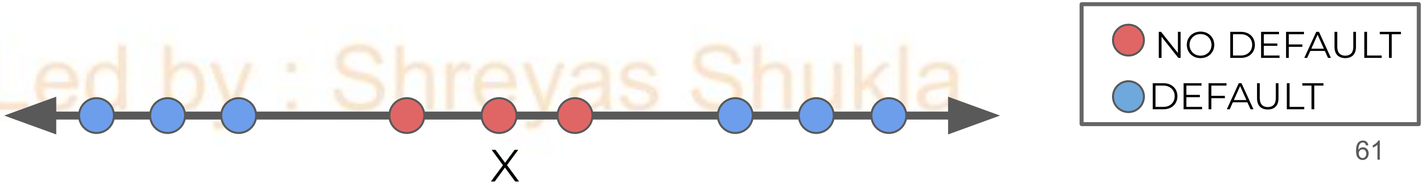
Recall our 1D example where classes were not easily separated by a single hyperplane:



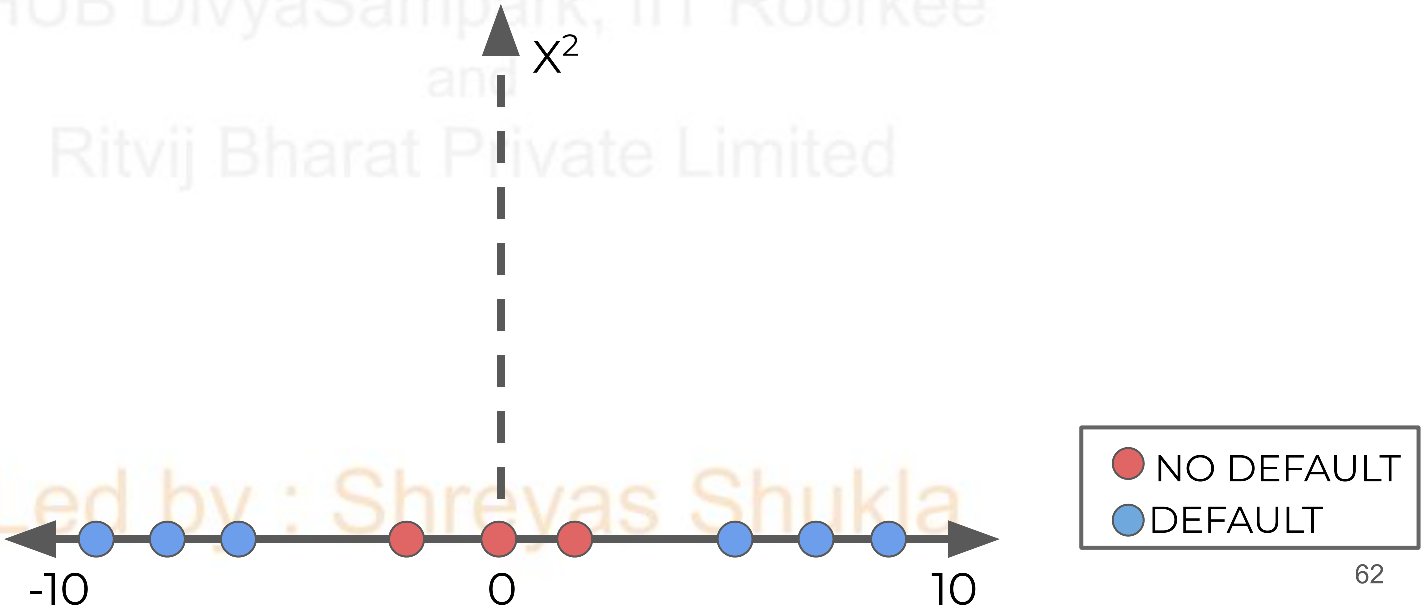
Let's explore how using a kernel could project this feature onto another dimension.



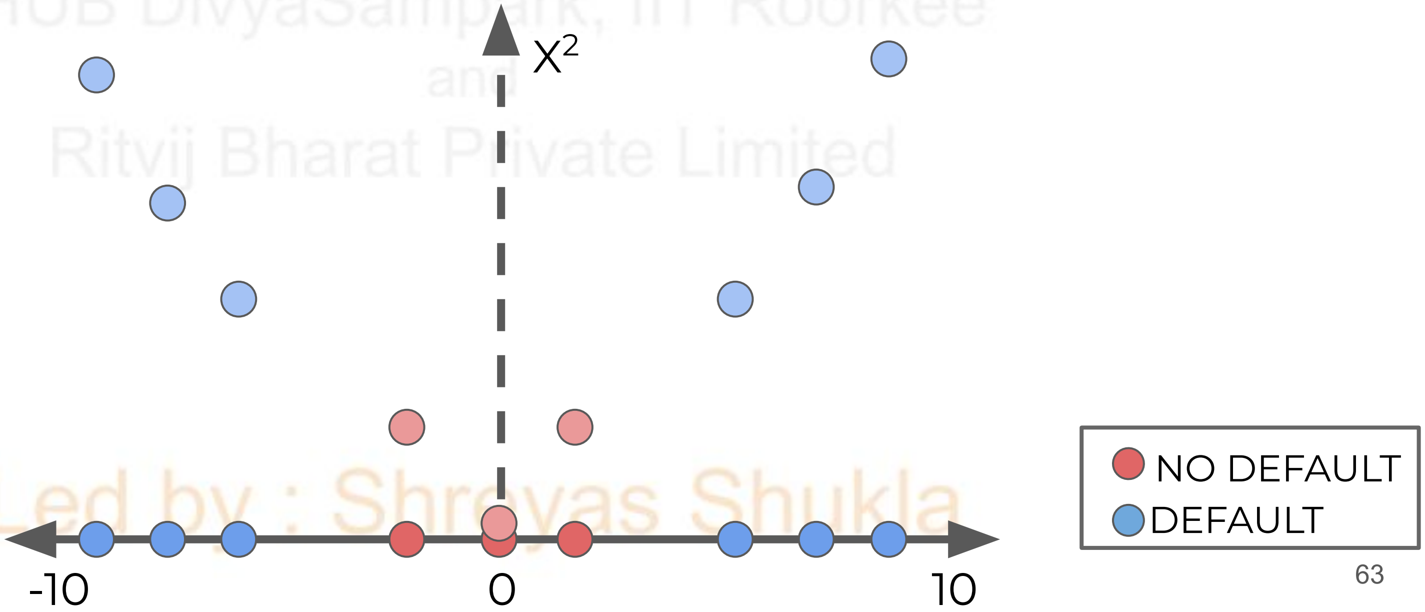
For example, a polynomial kernel could expand onto an X^2 dimension:



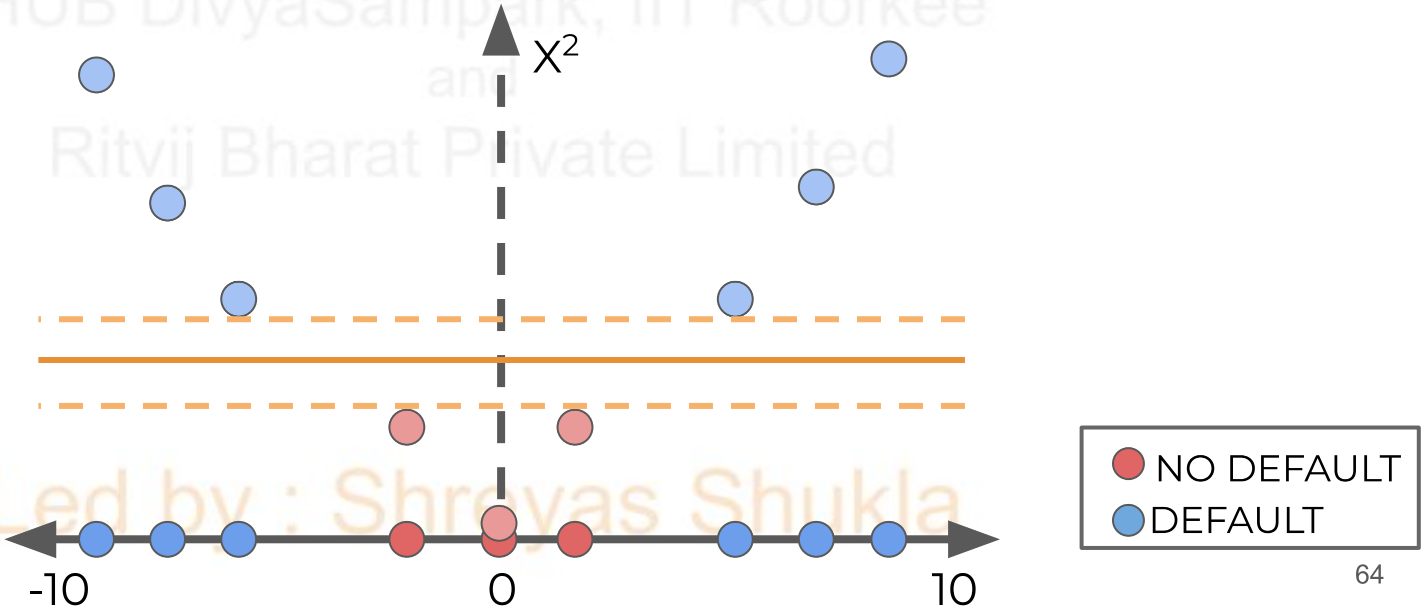
For example, a polynomial kernel could expand onto an X^2 dimension:



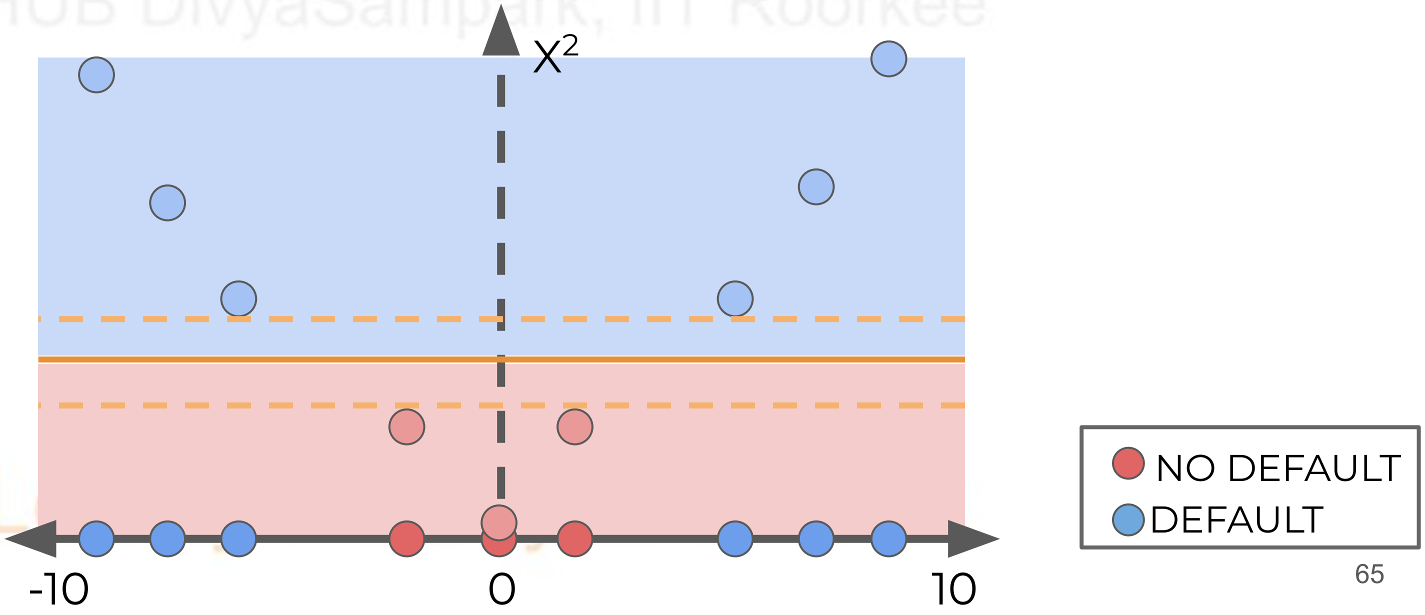
For example, a polynomial kernel could expand onto an X^2 dimension:



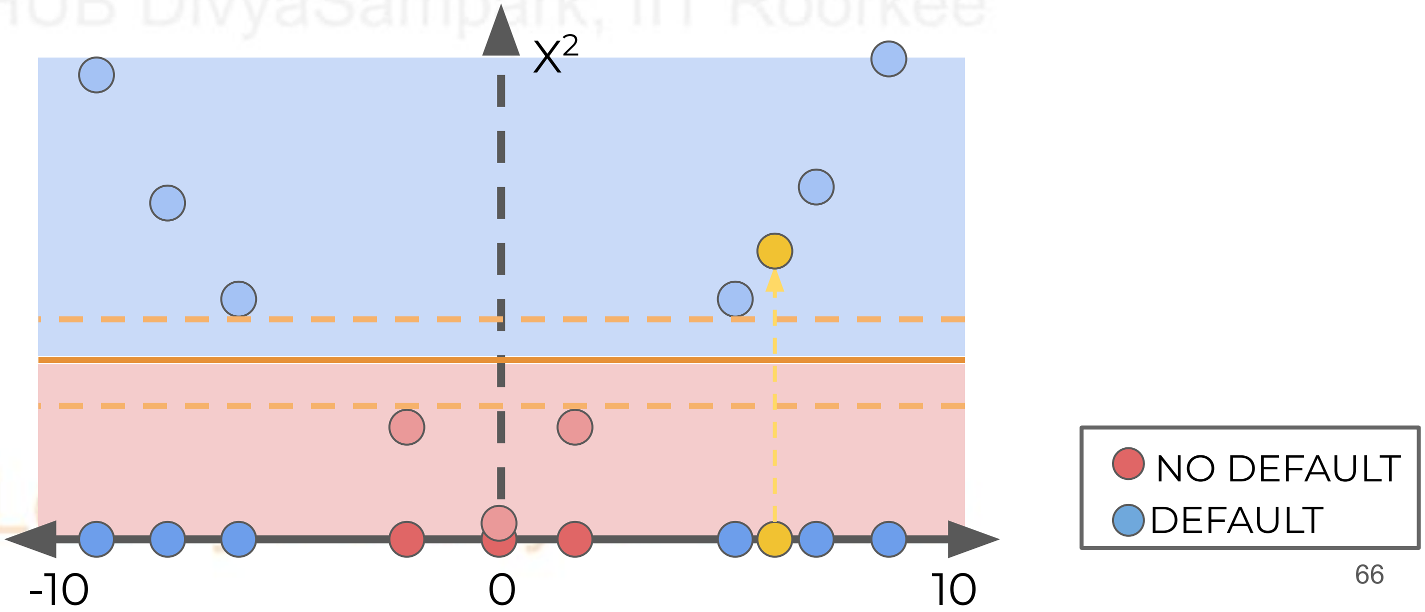
Create a hyperplane after projecting



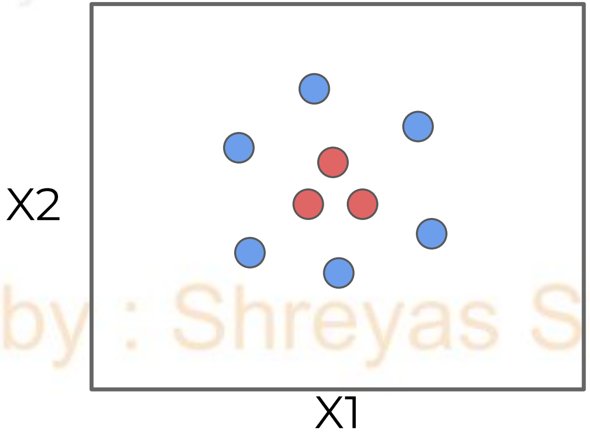
Create a hyperplane after this projection. Using this kernel projection, evaluate new points:



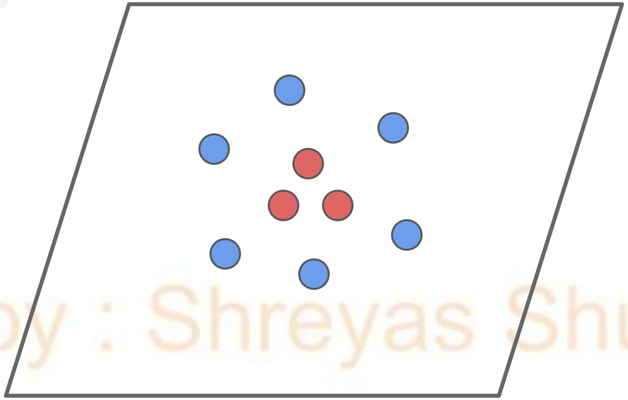
Evaluate new points



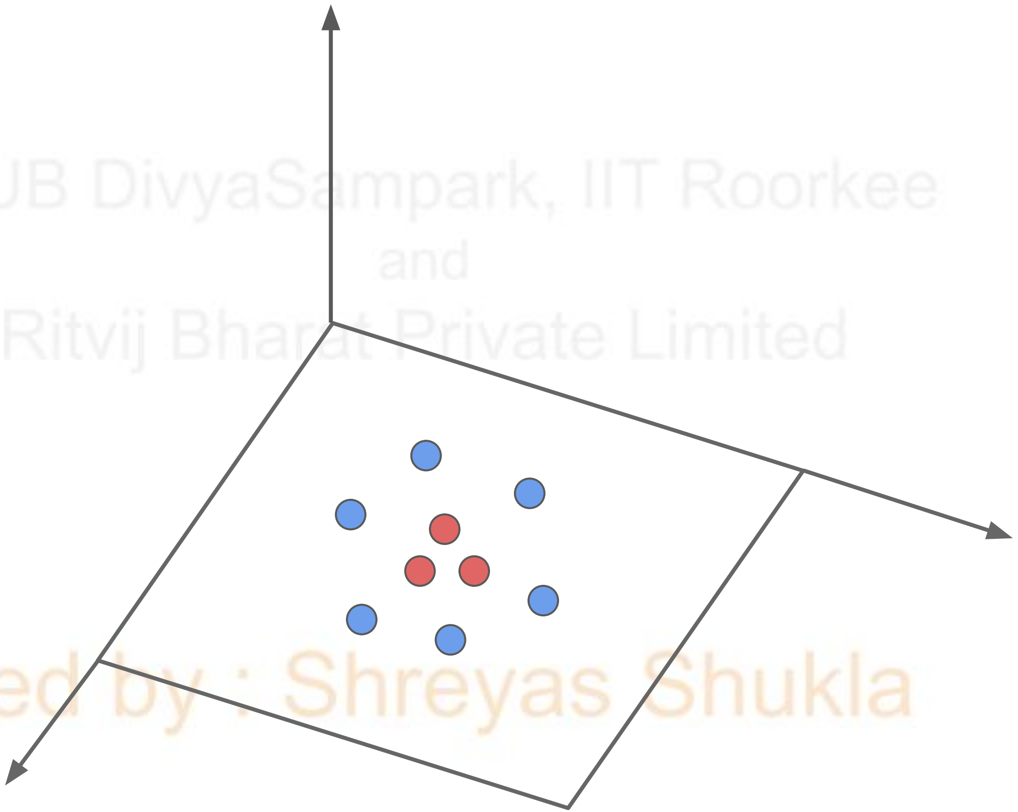
Now Imagine a 2D feature space where a hyperplane can not separate effectively, even with soft margins.



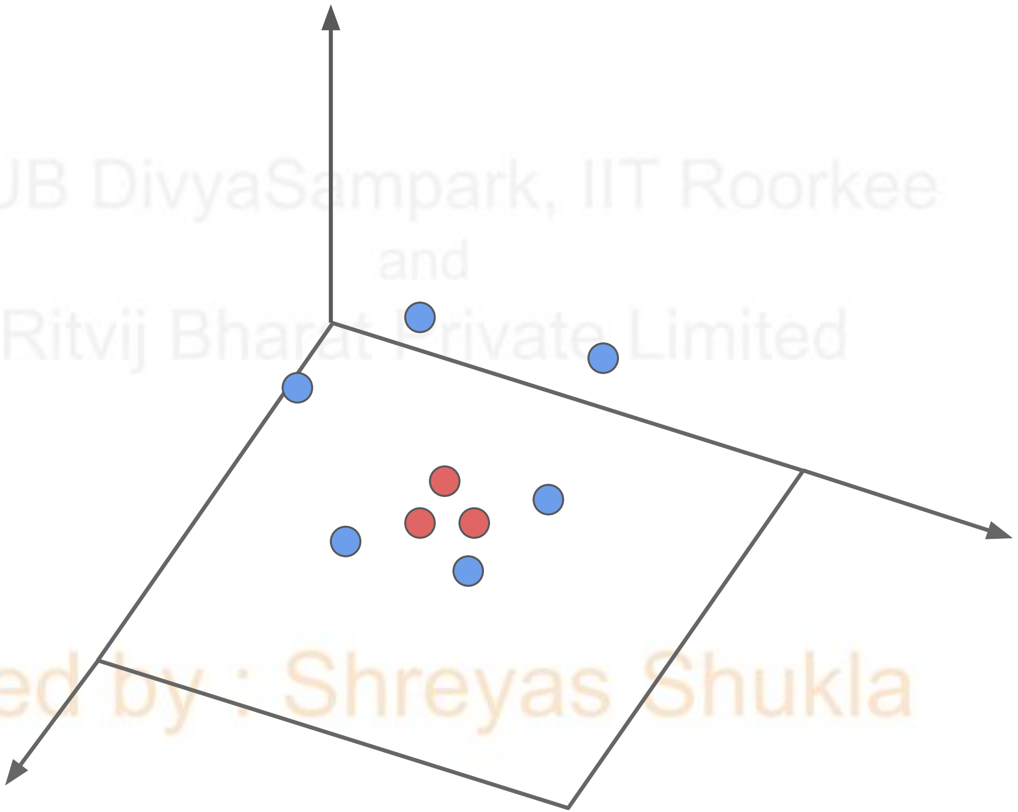
Here, we use SVMs to enable the use of a kernel transformation to project to a higher dimension.



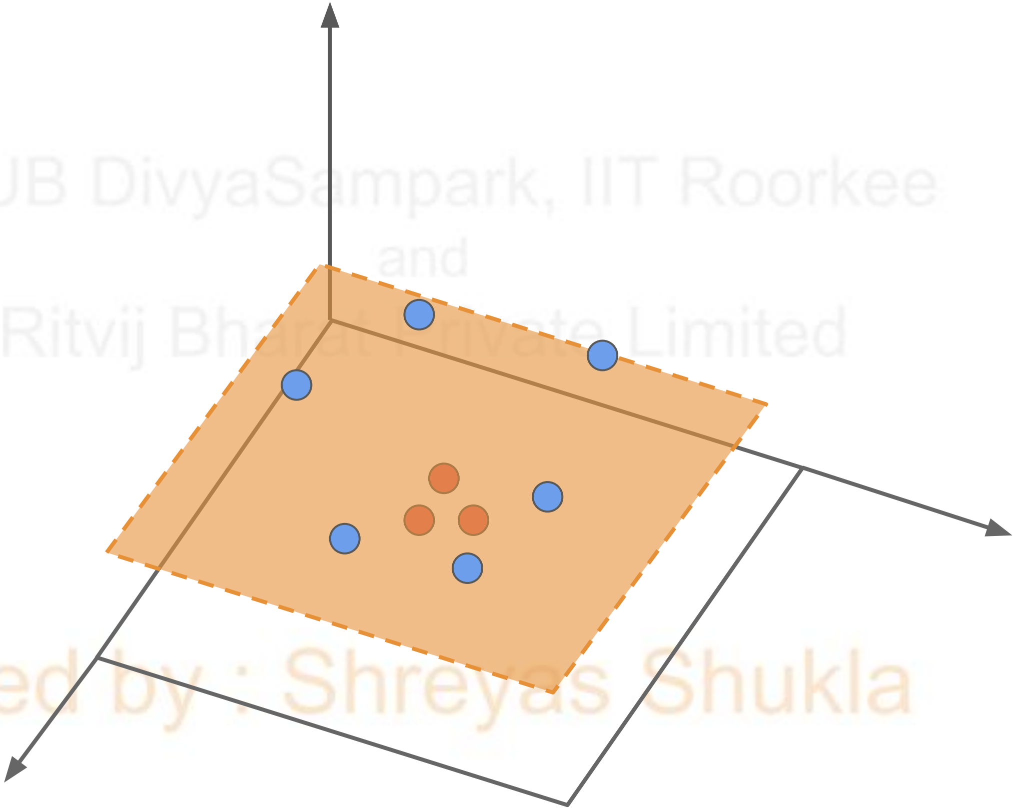
2D to 3D



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)!