Support Vector Machines

Table of Contents

[Classifier Margin 3](#_Toc101324728)

[Mathematical Interpretation 4](#_Toc101324729)

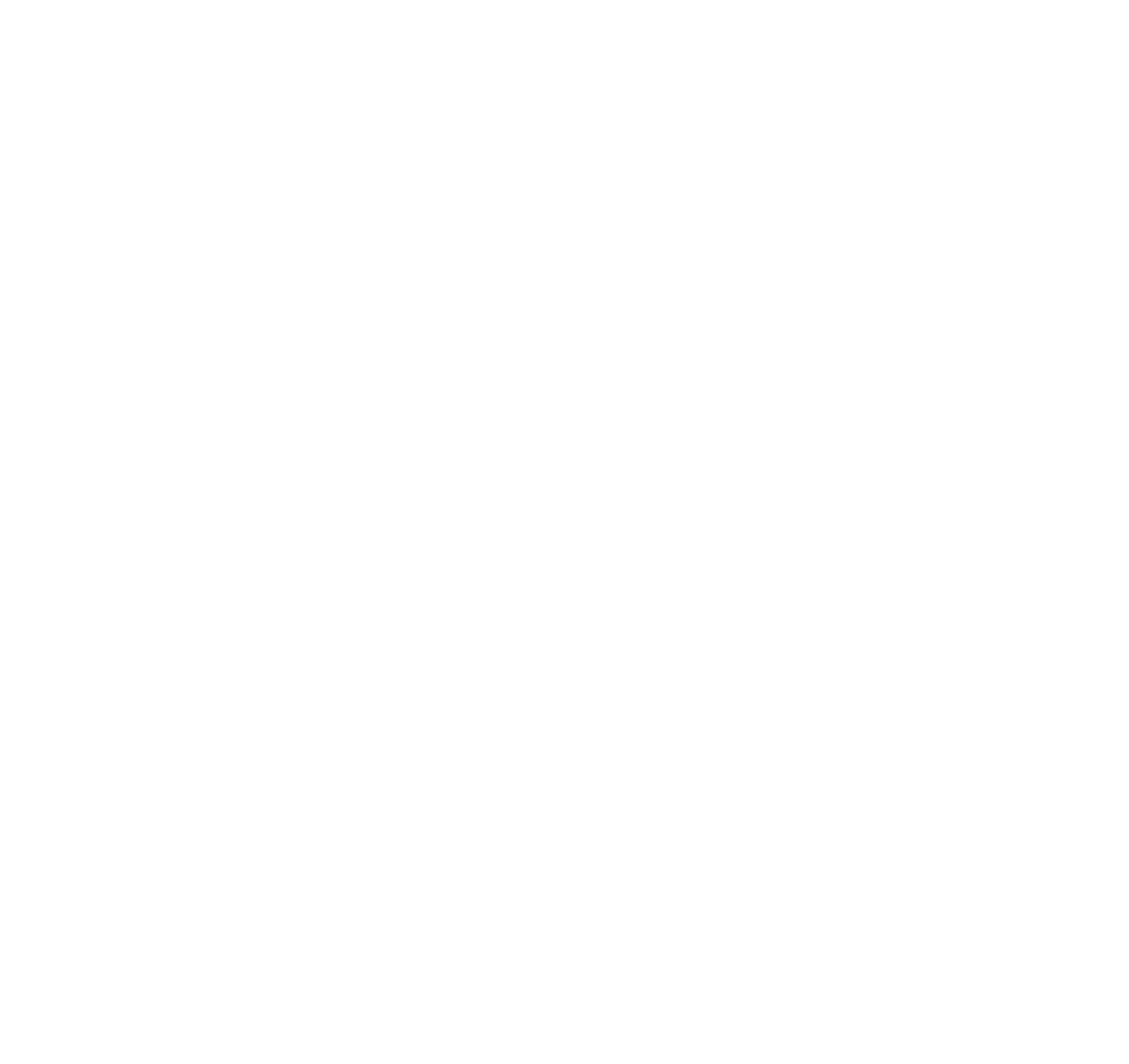
[Solving the Optimization Problem 5](#_Toc101324730)

[Lagrange Multipliers 6](#_Toc101324731)

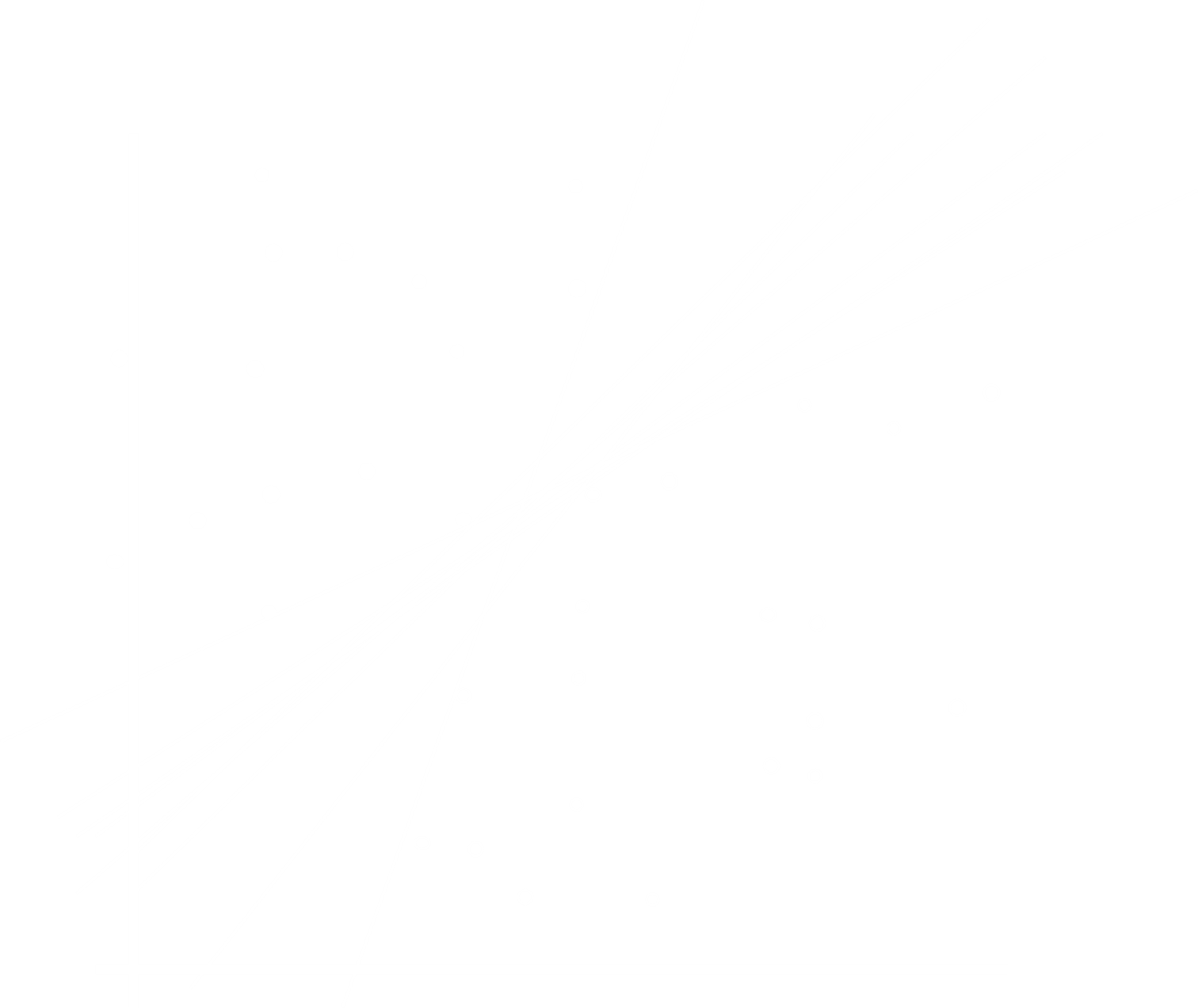
[Lagrange Multipliers with SVMs 7](#_Toc101324732)

[Non-Linear SVMs 8](#_Toc101324733)

[The Kernel Trick 9](#_Toc101324734)



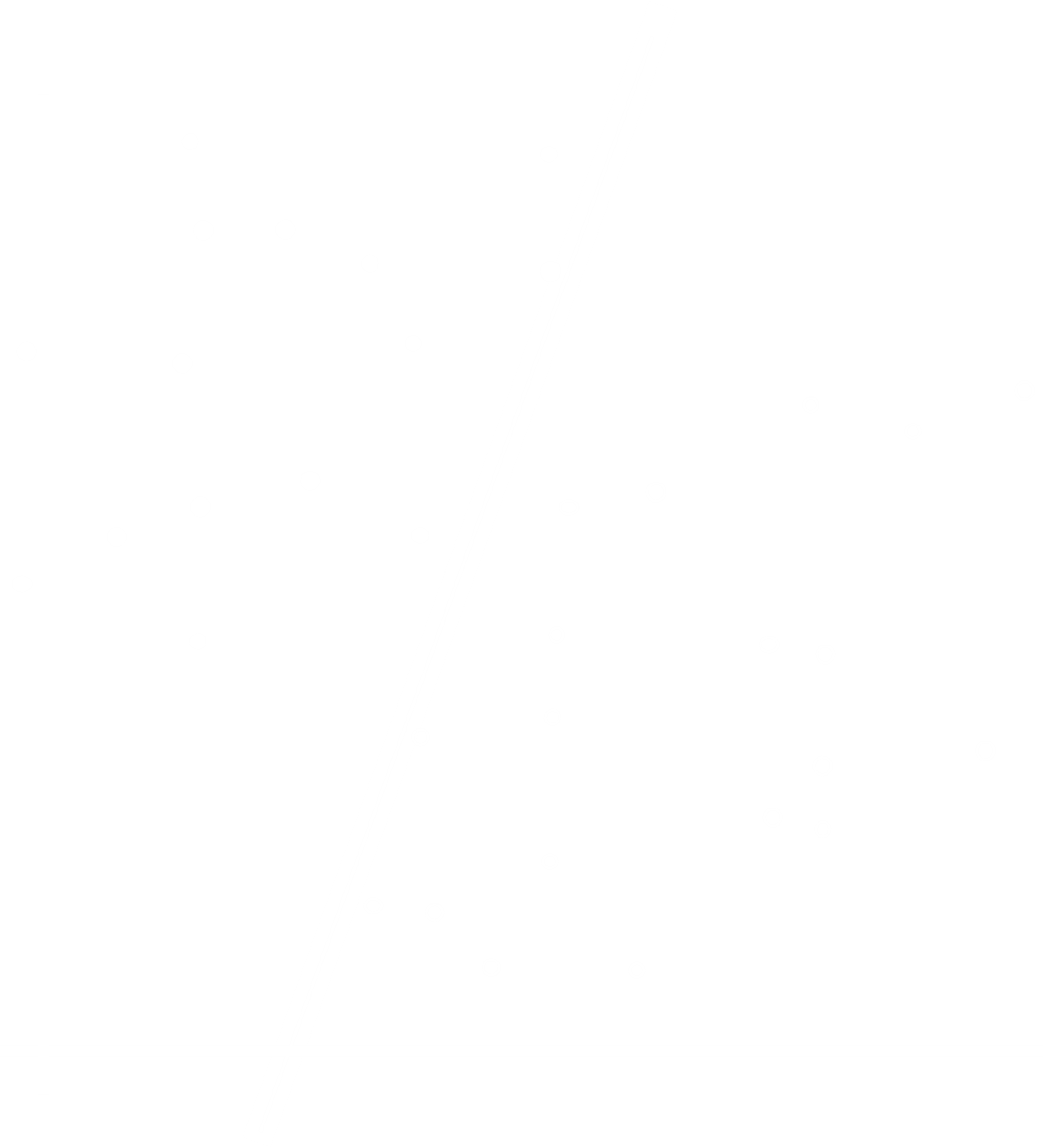
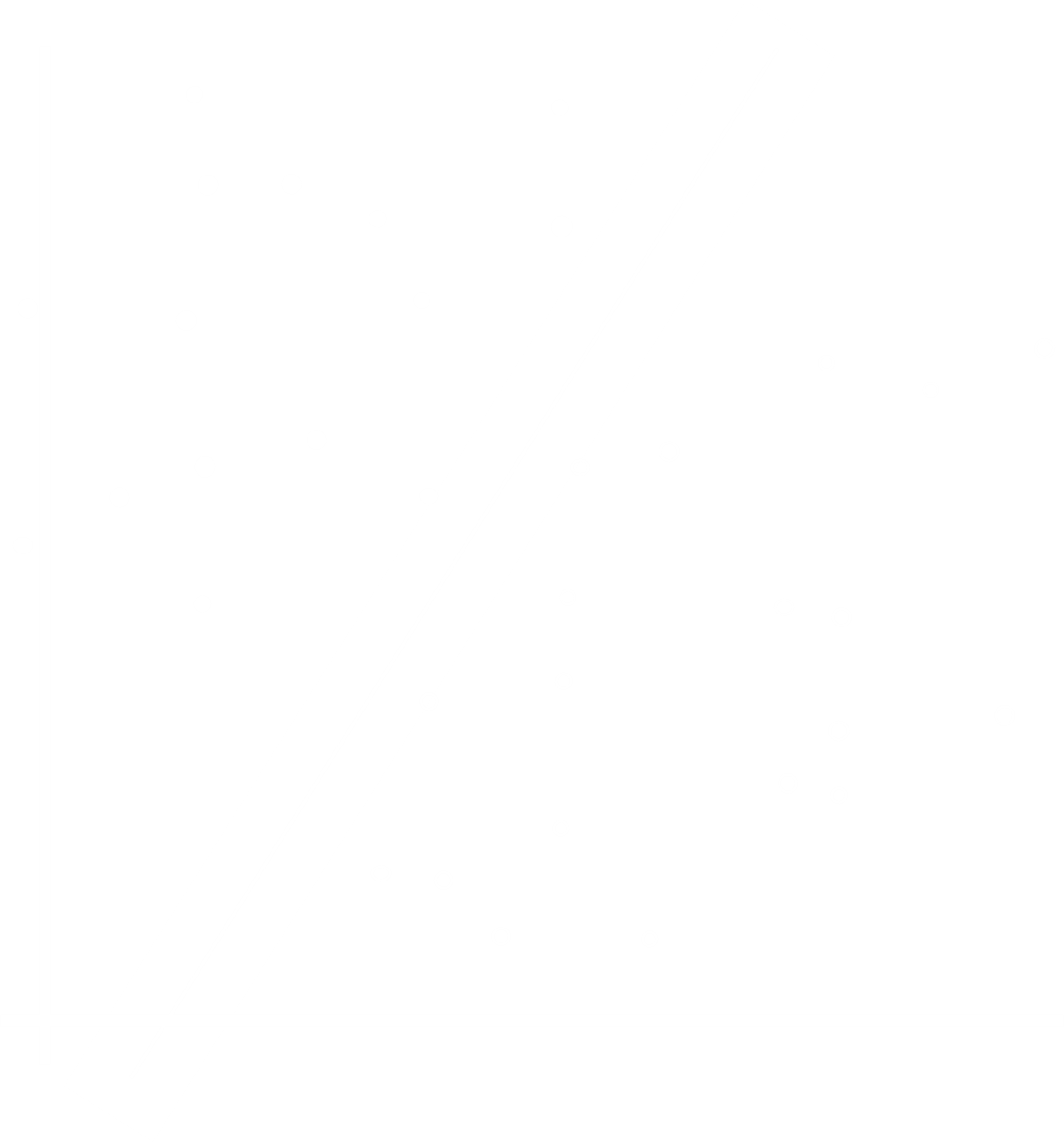
We have previously used **gradient descent** to find decision boundaries, such as the one above. The problem with this is, there can be many different lines which are valid choices, i.e. they divide the data correctly with no misclassifications.



In such a scenario, how do we choose which line is the best? The reason this question is important is, depending on which line we pick, **future data points** might be misclassified. This is the problem that **Support Vector Machines** (SVMs) attempt to solve.

## Classifier Margin

The **classifier margin** is defined as the width that the decision boundary could be increased by before it hits a data point. For example, the decision boundary on the left has a low margin, while the one on the right has a high margin.

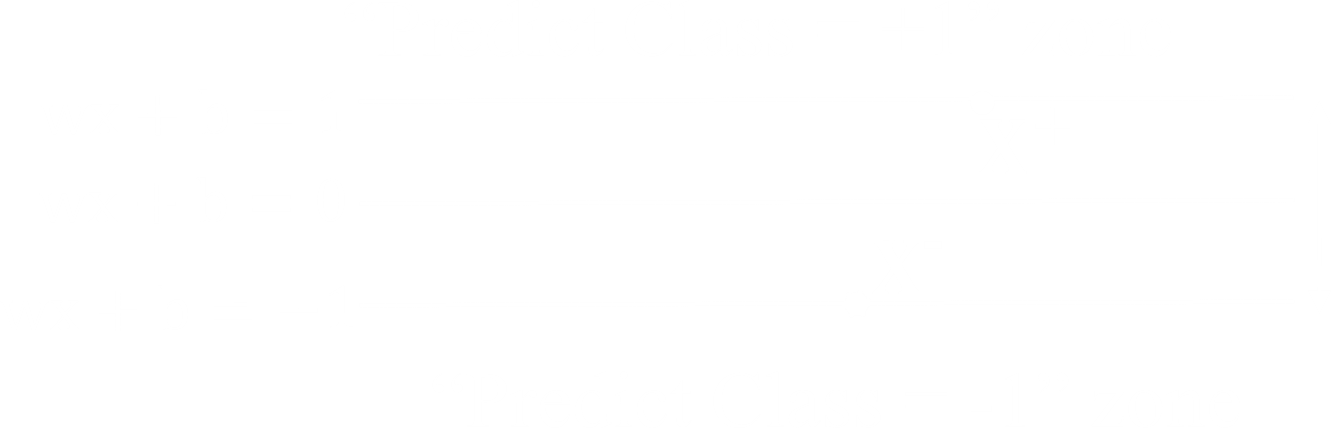
Intuitively we know that a **larger margin** is a better decision boundary. The specific type of SVM that finds the linear decision boundary with the maximum margin is called a **linear SVM**.

Note that the margin must be increased uniformly on both sides, i.e. a margin that is narrower on one side and wider on the other giving the same overall width does not count.

The data points at which the margin stops are called the **support vectors**.

## Mathematical Interpretation

The good part about linear SVMs is that we only need to worry about the support vectors. Consider that we have two lines parallel to the decision boundary, one going through the support vector on the positive side and the other going through the support vector on the negative side.



We cannot know for sure that these lines will have the values and respectively. However, if they have the values and , we can simply divide by to create this situation.

We have previously seen that the magnitude of the response for a line is given by . Since we want to find the distance between the two lines, which is the **margin**, it is given by:

For a given data point , if the correct label is , we know and if , then . This can be written as . Given this information, our goal is to maximize .

can also be written as . For simplicity, we can omit the square root (since the maximum value will still be the maximum value, regardless of the square root), which gives us . Another way of looking at this is that we want to minimize . We now have a **constrained optimization problem** in our hands.

## Solving the Optimization Problem

As a reminder, we must find values of and such that

1. is minimized

This is a **constrained optimization problem**, since we are trying to optimize one equation while being constrained by another. In our case, the constrain applies for different samples, so we have constraints. One way of solving such problems is to use the **Lagrange multiplier**.

### Lagrange Multipliers

To understand how Lagrange multiplies work, consider that we are trying to maximize the equation , under the condition . To solve this, we first put the equations into the **Lagrangian form**:

We can now solve this:

Solving these equations simultaneously, we get , and . The problem we were originally tyring to solve is to find the value of , which is .

### Lagrange Multipliers with SVMs

Coming back to the SVM problem, the equation is said to be in its **primal form**. We convert this to its **dual form**. Since we have constraints, we will also have different Lagrange multipliers (i.e. we are trying to find ). Without going into the details, this can be brought down to:

1. Maximize

The fun part is that for **non-support vectors**, the values will be , while for support vectors, .

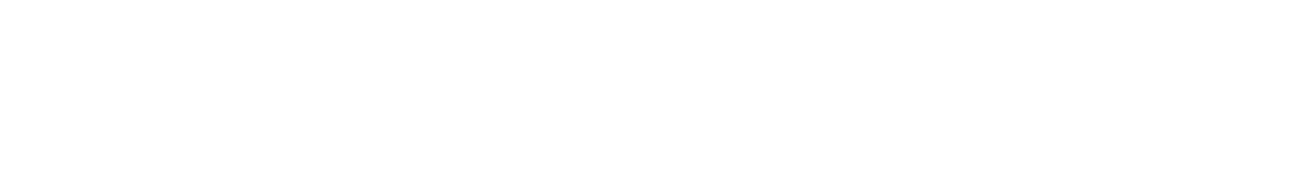
From this, we can derive that

for any such that

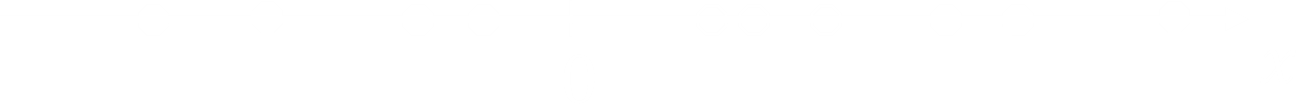
For new data points, , we compute as

## Non-Linear SVMs

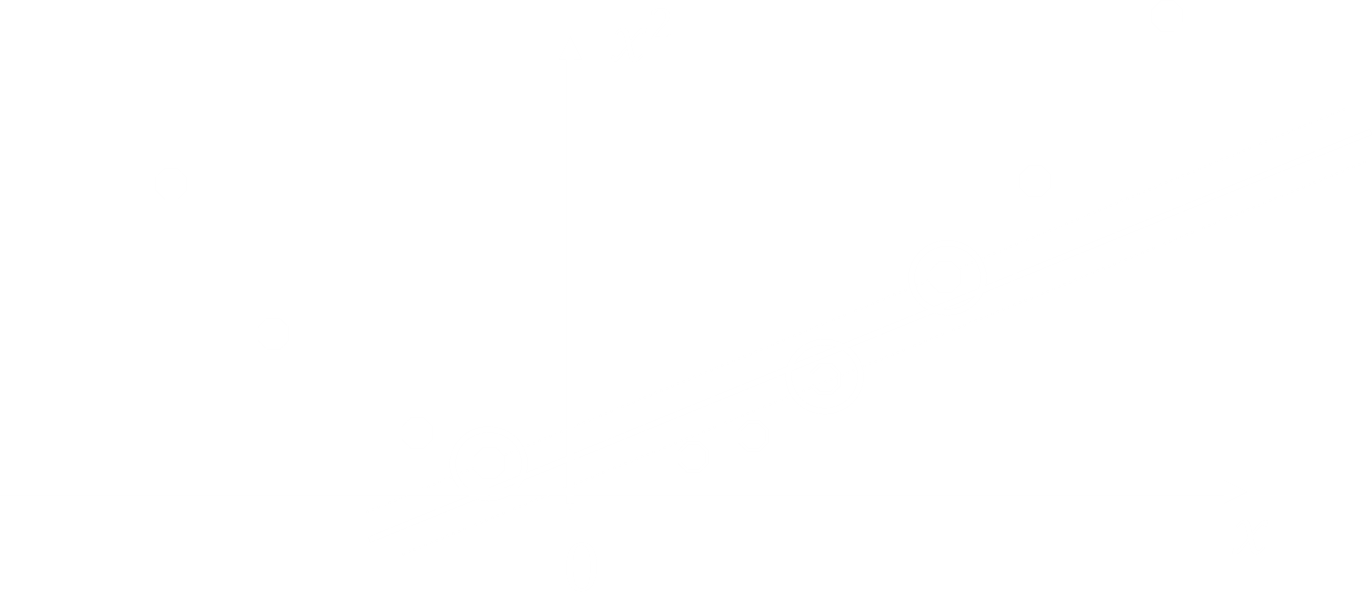
There are some datasets that are very easy to separate linearly.



But not all datasets are so easy.

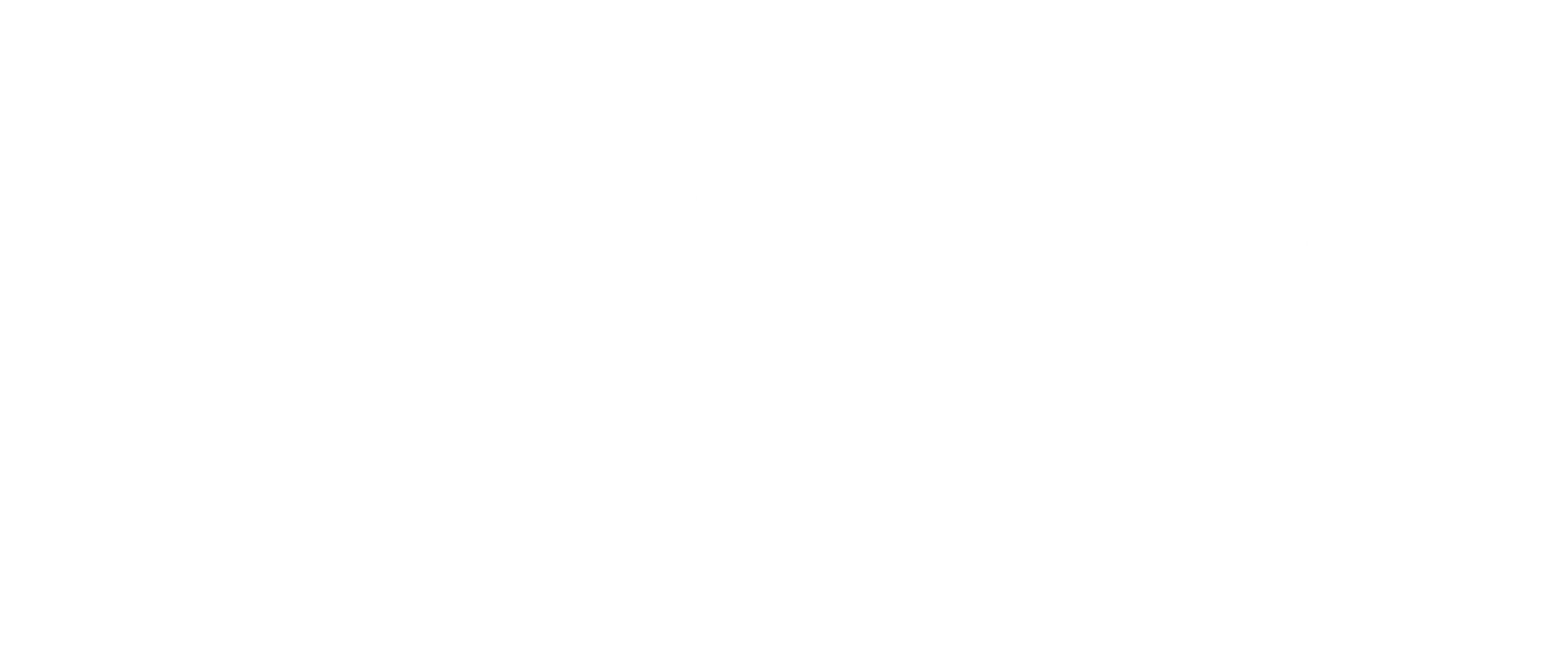


For datasets like this, it might be easier for us to find a linear decision boundary if we map the dataset onto a **higher dimensional plane**.



This mapping has been done by simply using as the -axis. By doing this mapping, we are now able to find a linear decision boundary.

The original feature space can always be converted to a higher-dimensional one in which a linear decision boundary can be found.



The linear SVM we have found is actually non-linear in the original feature space.

### The Kernel Trick

For a new data point , we previously saw that we can calculate as . As can be seen, the output depends on the **dot product** of two vectors, . If we map each data point onto a higher-dimension, i.e. , then we get .

It is expensive to map all the datapoints to a higher dimension space, find the SVM and then bring the SVM back to the original space. A **kernel function** gives us the same value in both spaces.

For a 2-dimensional vector , let . We need to show that .

There are several other kernel functions like this, such as:

* Linear -
* Polynomial -
* Gaussian -
* Sigmoid -