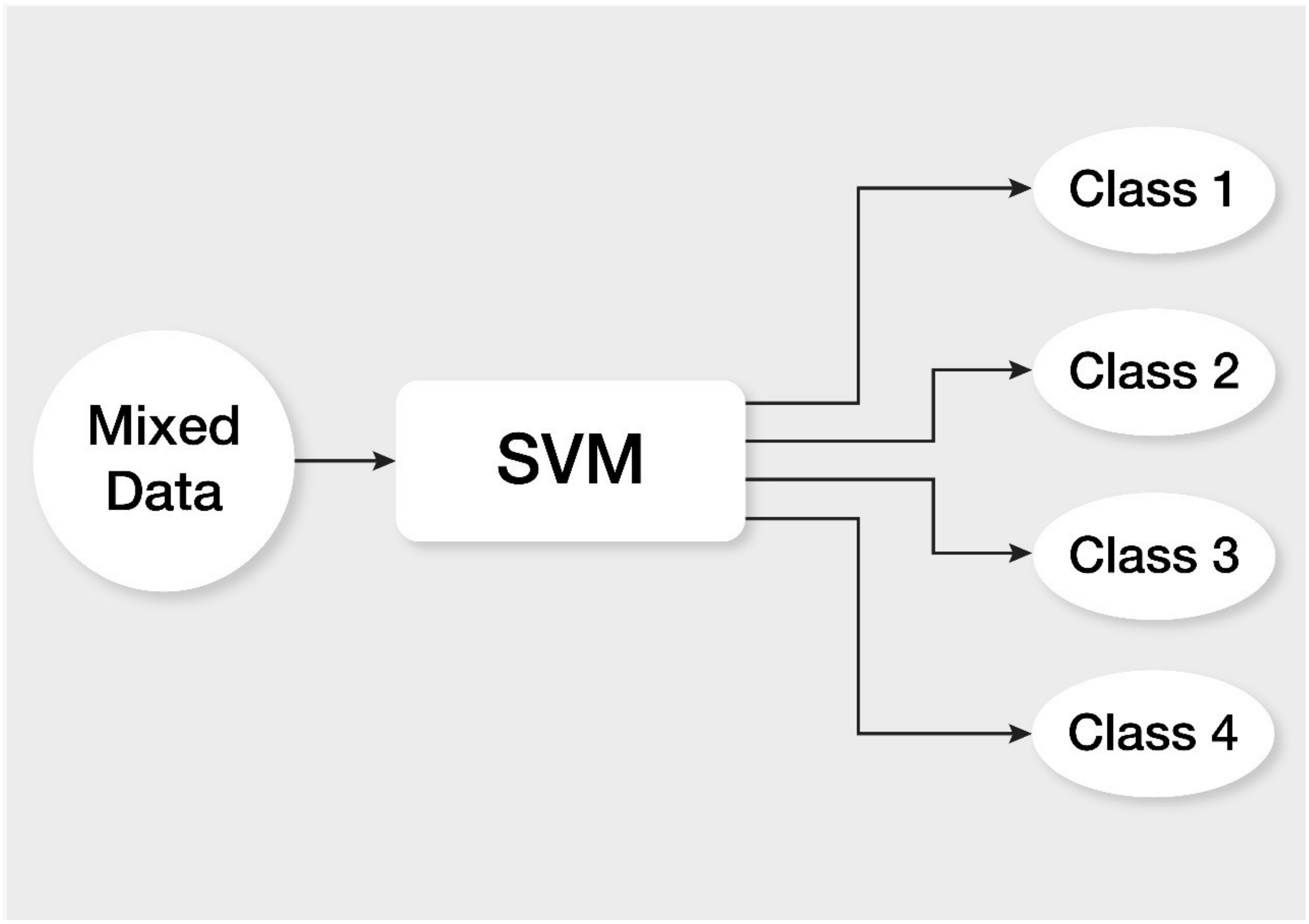
Support Vector Machines(SVM) — An Overview

Rushikesh Pupale Follow
Jun 16, 2018 · 7 min read



SVM classifier

Machine learning involves predicting and classifying data and to do so we employ various machine learning algorithms according to the dataset.

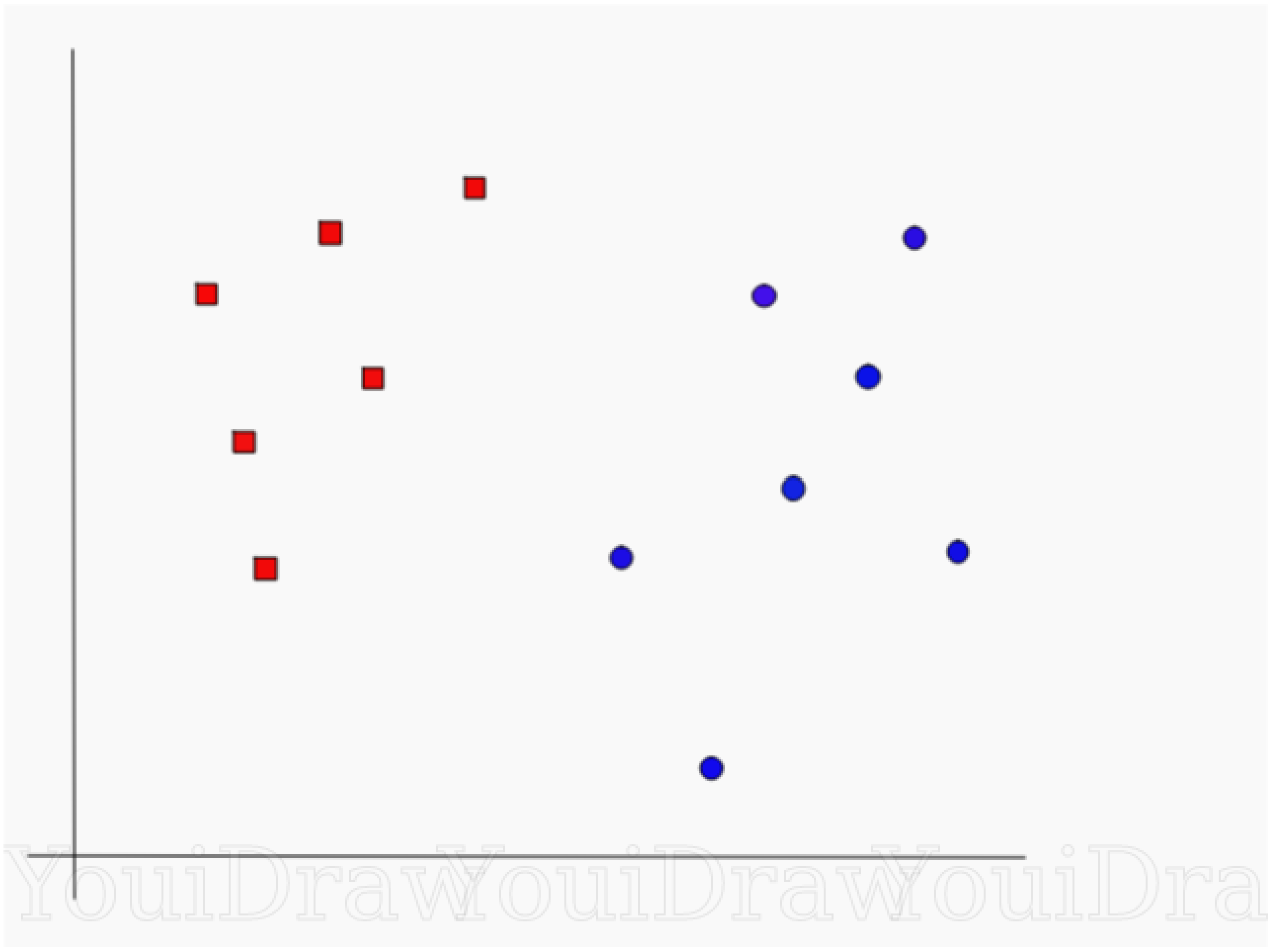
SVM or Support Vector Machine is a linear model for classification and regression problems. It can solve linear and non-linear problems and work well for many practical problems. The idea of SVM is simple: The algorithm creates a line or a hyperplane which separates the data into classes.

In this blog post I plan on offering a high-level overview of SVMs. I will talk about the theory behind SVMs, it's application for non-linearly separable datasets and a quick example of implementation of SVMs in Python as well. In the upcoming articles I will explore the maths behind the algorithm and dig under the hood.

THEORY

At first approximation what SVMs do is to find a separating line(or hyperplane) between data of two classes. SVM is an algorithm that takes the data as an input and outputs a line that separates those classes if possible.

Lets begin with a problem. Suppose you have a dataset as shown below and you need to classify the red rectangles from the blue ellipses(let's say positives from the negatives). So your task is to find an ideal line that separates this dataset in two classes (say red and blue).

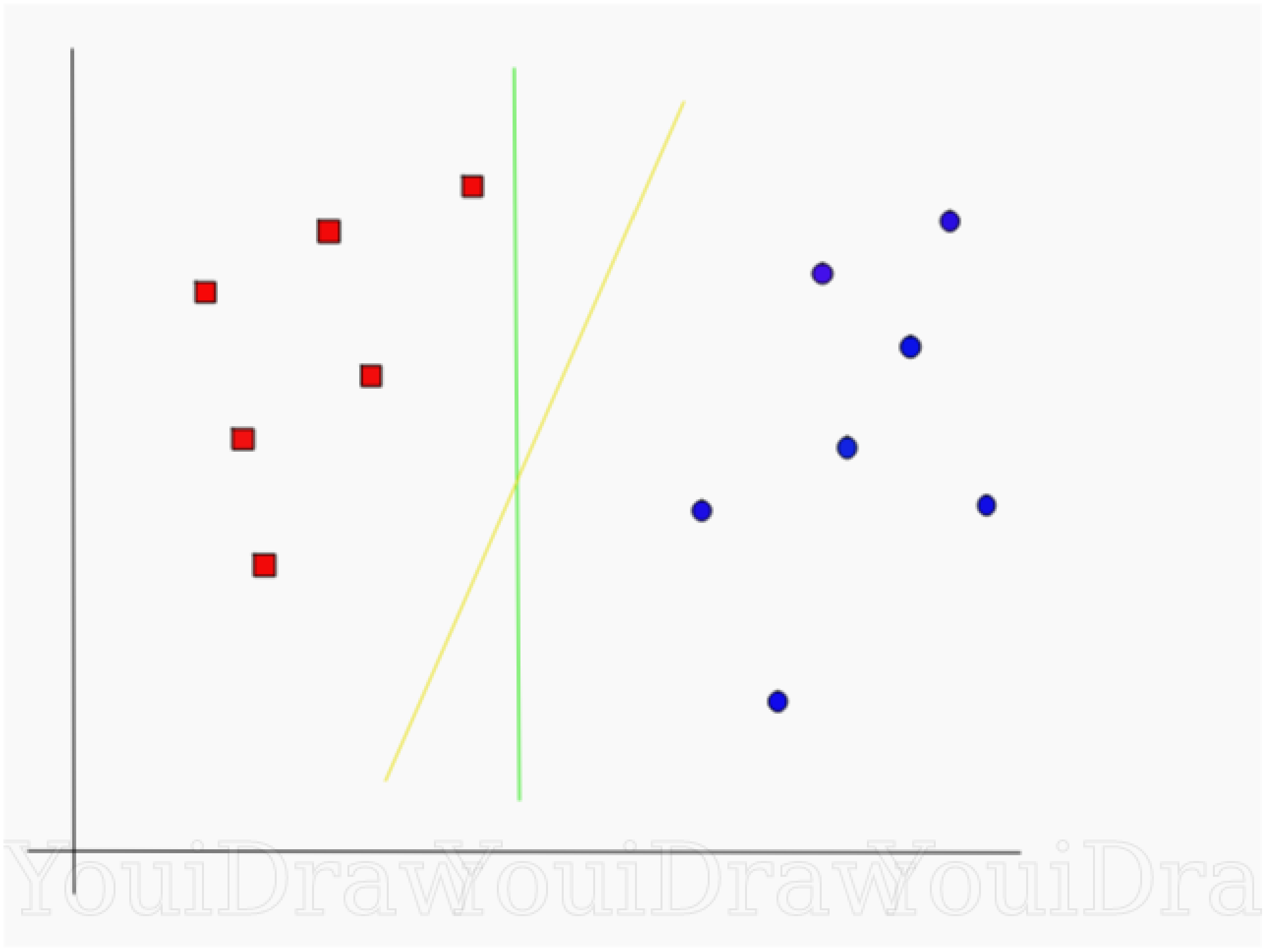


Find an ideal line/ hyperplane that separates this dataset into red and blue categories

Not a big task, right?

But, as you notice there isn't a unique line that does the job. In fact, we have an infinite lines that can separate these two classes. So how does SVM find the ideal one???

Let's take some probable candidates and figure it out ourselves.



Which line according to you best separates the data???

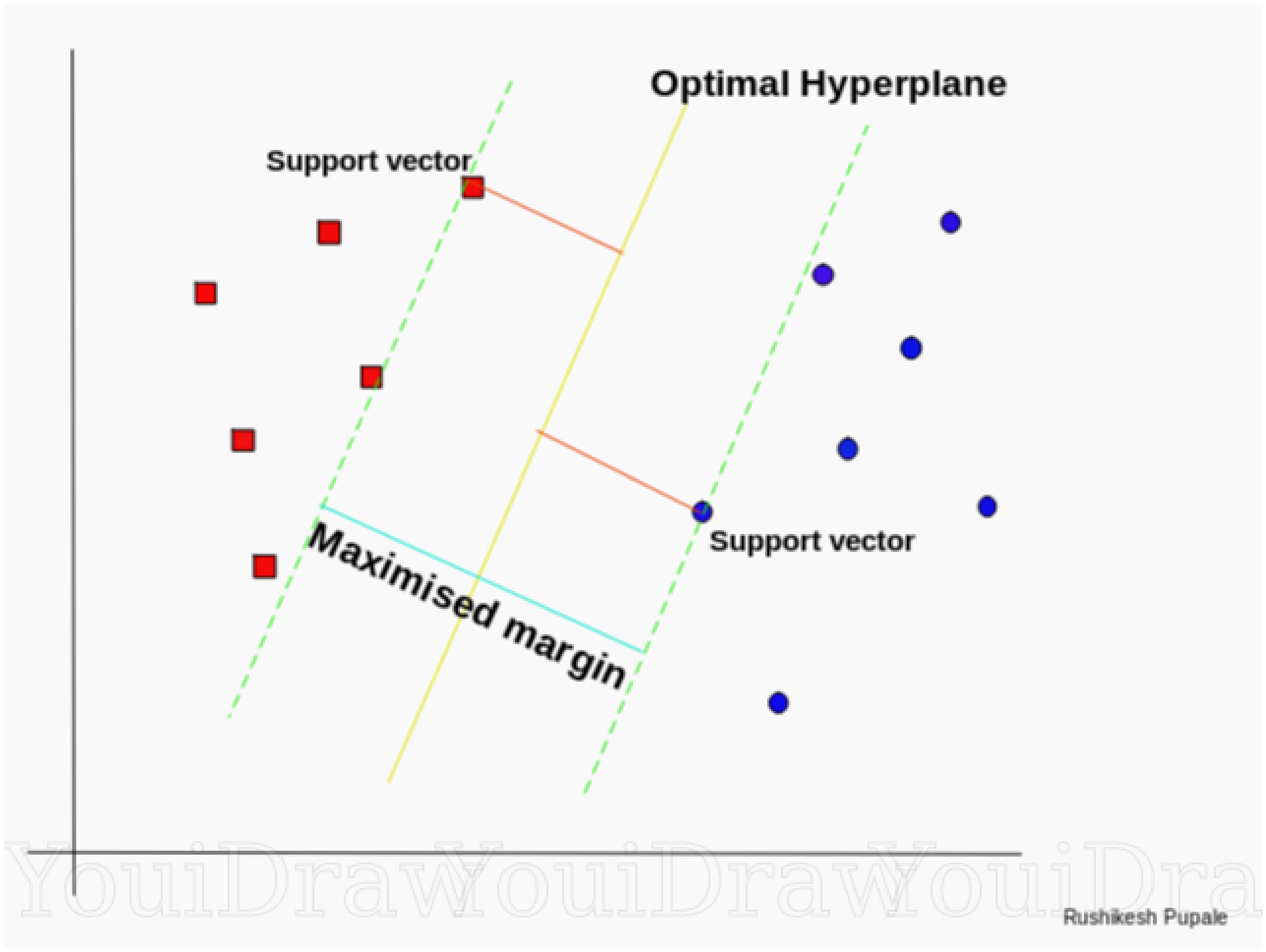
We have two candidates here, the green colored line and the yellow colored line. Which line according to you best separates the data?

If you selected the yellow line then congrats, because thats the line we are looking for. It's visually quite intuitive in this case that the yellow line classifies better. But, we need something concrete to fix our line.

The green line in the image above is quite close to the red class. Though it classifies the current datasets it is not a generalized line and in machine learning our goal is to get a more generalized separator.

SVM’s way to find the best line

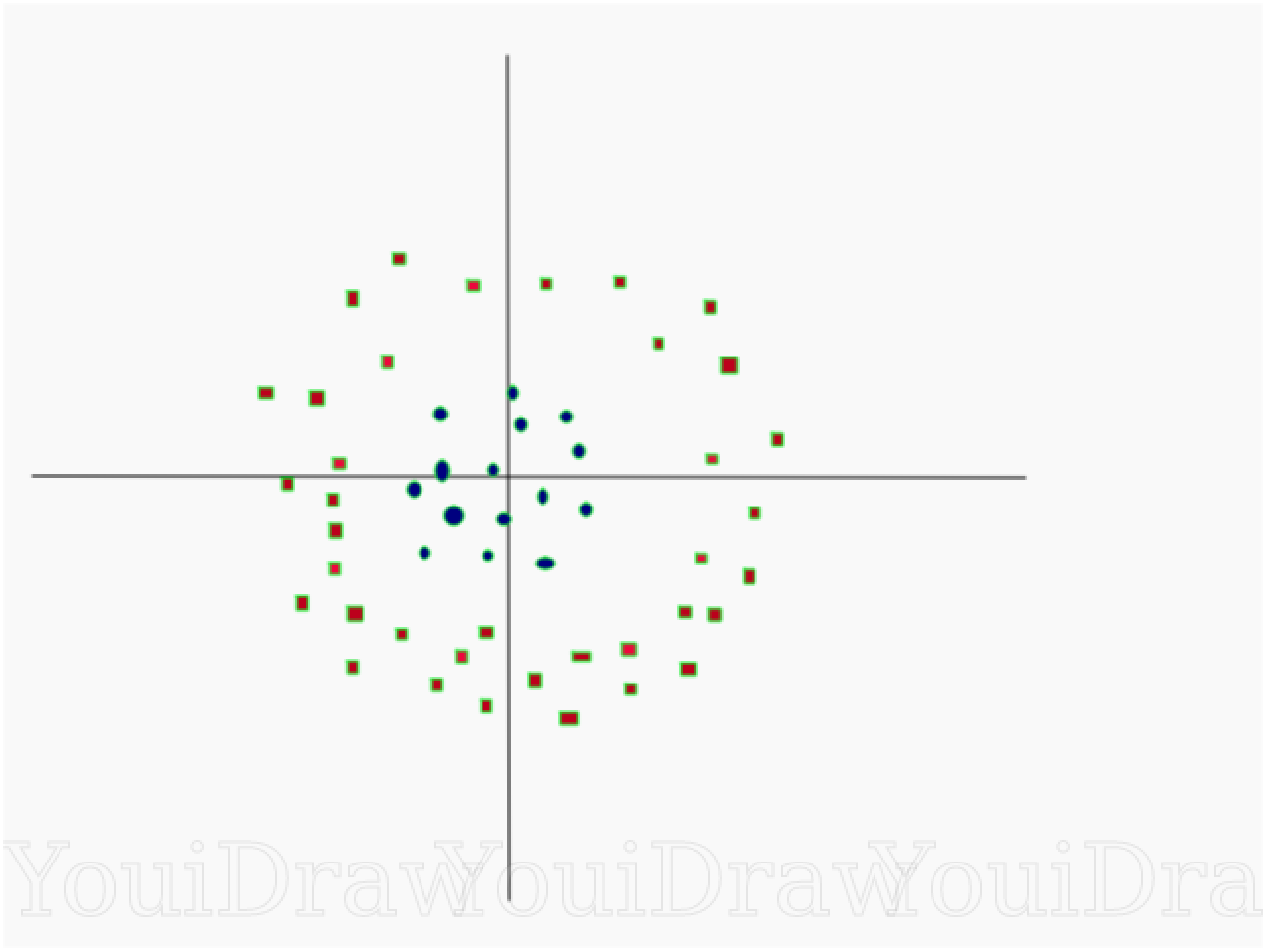
According to the SVM algorithm we find the points closest to the line from both the classes.These points are called support vectors. Now, we compute the distance between the line and the support vectors. This distance is called the margin. Our goal is to maximize the margin. The hyperplane for which the margin is maximum is the optimal hyperplane.



Optimal Hyperplane using the SVM algorithm

Thus SVM tries to make a decision boundary in such a way that the separation between the two classes(that street) is as wide as possible.

Simple, ain’t it? Let’s consider a bit complex dataset, which is not linearly separable.

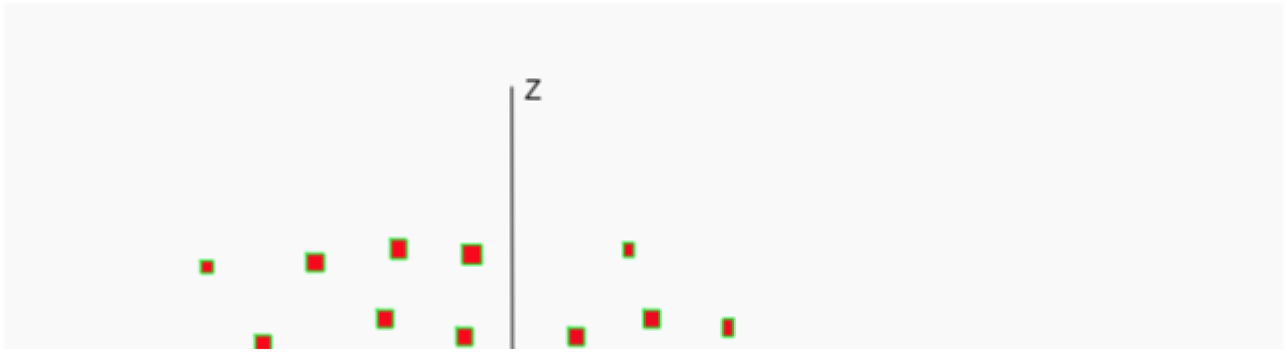


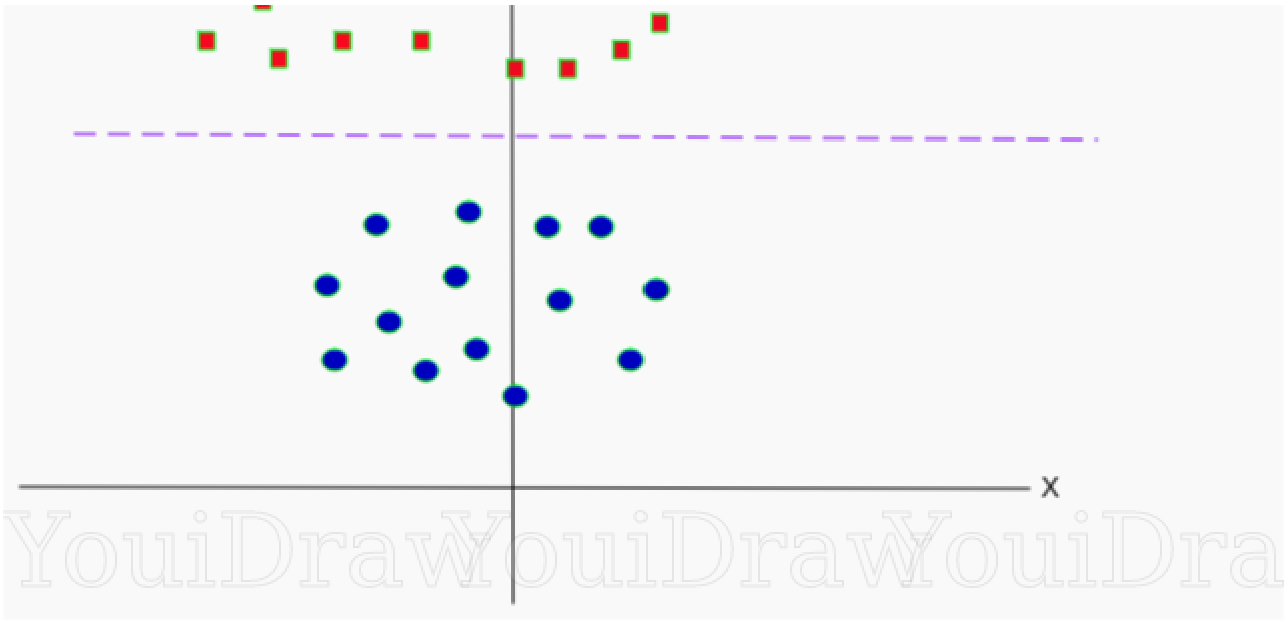
Non-linearly separable data

This data is clearly not linearly separable. We cannot draw a straight line that can classify this data. But, this data can be converted to linearly separable data in higher dimension. Lets add one more dimension and call it z-axis. Let the co-ordinates on z-axis be governed by the constraint,

$z = x^2+y^2$

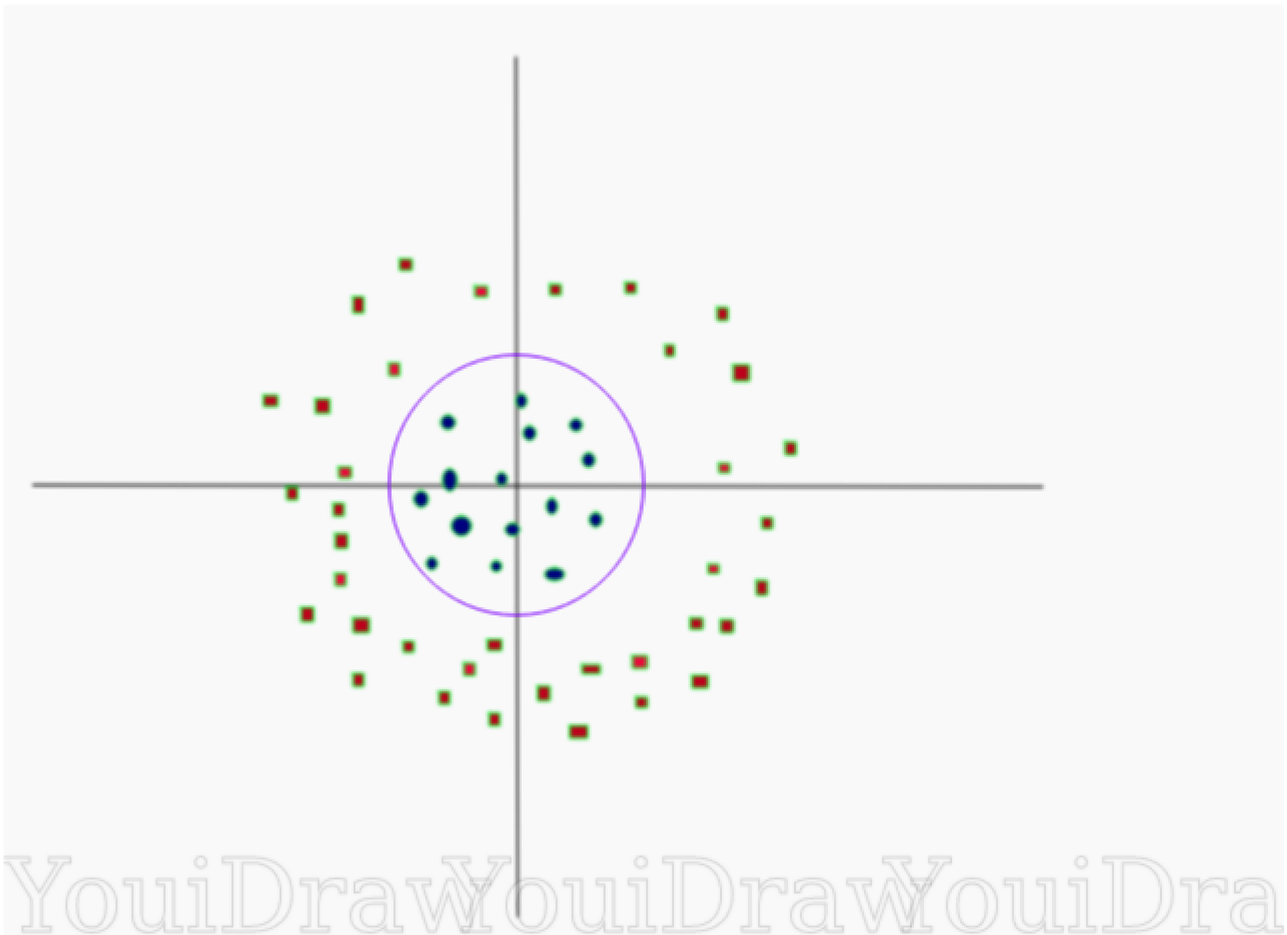
So, basically z co-ordinate is the square of distance of the point from origin. Let's plot the data on z-axis.





Dataset on higher dimension

Now the data is clearly linearly separable. Let the purple line separating the data in higher dimension be z=k, where k is a constant. Since, $z=x^2+y^2$ we get $x^2 + y^2 = k$; which is an equation of a circle. So, we can project this linear separator in higher dimension back in original dimensions using this transformation.



Decision boundary in original dimensions

Thus we can classify data by adding an extra dimension to it so that it becomes linearly separable and then projecting the decision boundary back to original dimensions using mathematical transformation. But finding the correct transformation for any given dataset isn't that easy. Thankfully, we can use kernels in sklearn's SVM implementation to do this job.

HYPERPLANE

Now that we understand the SVM logic lets formally define the hyperplane .

> ***A hyperplane in an n-dimensional Euclidean space is a flat, n-1 dimensional subset of that space that divides the space into two disconnected parts.***

For example let's assume a line to be our one dimensional Euclidean space(i.e. let's say our datasets lie on a line). Now pick a point on the line, this point divides the line into two parts. The line has 1 dimension, while the point has 0 dimensions. So a point is a hyperplane of the line.

For two dimensions we saw that the separating line was the hyperplane. Similarly, for three dimensions a plane with two dimensions divides the 3d space into two parts and thus act as a hyperplane. Thus for a space of n dimensions we have a hyperplane of n-1 dimensions separating it into two parts

CODE

```
import numpy as np
X = np.array([[-1, -1], [-2, -1], [1, 1], [2, 1]])
y = np.array([1, 1, 2, 2])
```

We have our points in X and the classes they belong to in Y.

Now we train our SVM model with the above dataset.For this example I have used a linear kernel.

```
from sklearn.svm import SVC
clf = SVC(kernel='linear')
clf.fit(X, y)
```

To predict the class of new dataset

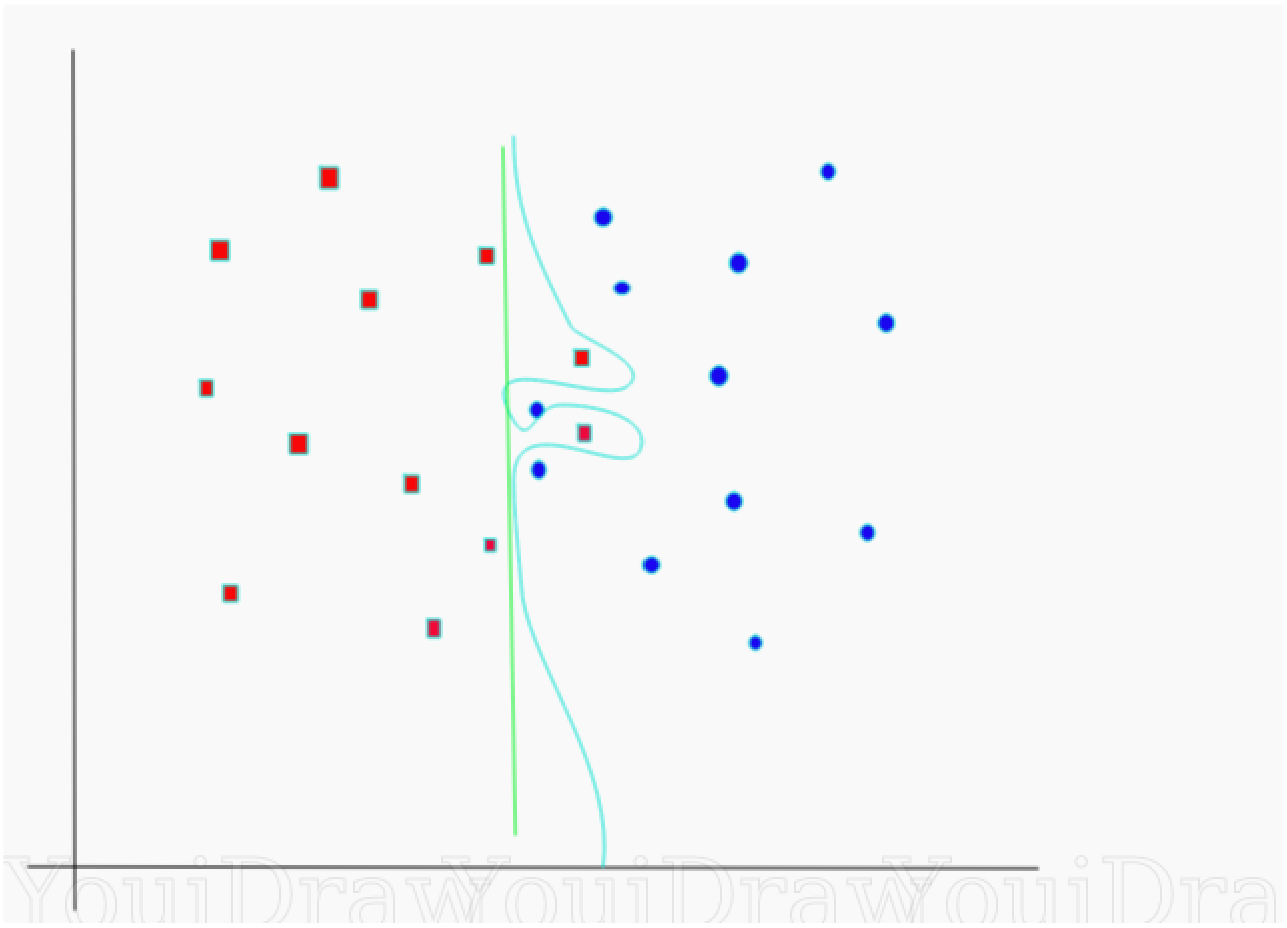
```
prediction = clf.predict([[0,6]])
```

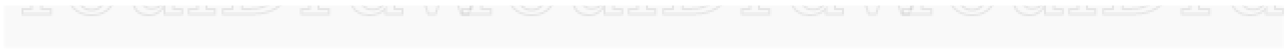
TUNING PARAMETERS

Parameters are arguments that you pass when you create your classifier. Following are the important parameters for SVM-

1]C:

It controls the trade off between smooth decision boundary and classifying training points correctly. A large value of c means you will get more training points correctly.





Smooth decision boundary vs classifying all points correctly

Consider an example as shown in the figure above. There are a number of decision boundaries that we can draw for this dataset. Consider a straight (green colored) decision boundary which is quite simple but it comes at the cost of a few points being misclassified. These misclassified points are called outliers. We can also make something that is considerably more wiggly(sky blue colored decision boundary) but where we get potentially all of the training points correct. Of course the trade off having something that is very intricate, very complicated like this is that chances are it is not going to generalize quite as well to our test set. So something that is simple, more straight maybe actually the better choice if you look at the accuracy. Large value of c means you will get more intricate decision curves trying to fit in all the points. Figuring out how much you want to have a smooth decision boundary vs one that gets things correct is part of artistry of machine learning. So try different values of c for your dataset to get the perfectly balanced curve and avoid over fitting.

2]Gamma:

It defines how far the influence of a single training example reaches. If it has a low value it means that every point has a far reach and conversely high value of gamma means that every point has close reach.

If gamma has a very high value, then the decision boundary is just going to be dependent upon the points that are very close to the line which effectively results in ignoring some of the points that are very far from the decision boundary. This is because the closer points get more weight and it results in a wiggly curve as shown in previous graph.On the other hand, if the gamma value is low even the far away points get considerable weight and we get a more linear curve.

In the upcoming article

I will explore the math behind the SVM algorithm and the optimization problem.

Conclusion

I hope this blog post helped in understanding SVMs. ***Comment down your thoughts, feedback or suggestions*** if any below.

Machine Learning Data Science Algorithms AI Artificial Intelligence