AI: Support Vector Machines

Author: Joel Fernandes <joel@joelfernandes.org> Last updated: June 25, 2023.

Before reading this article, make sure to read my other article on "AI: Vector Basics", as an understanding of vectors is a prerequisite.

Weights and biases of an SVM

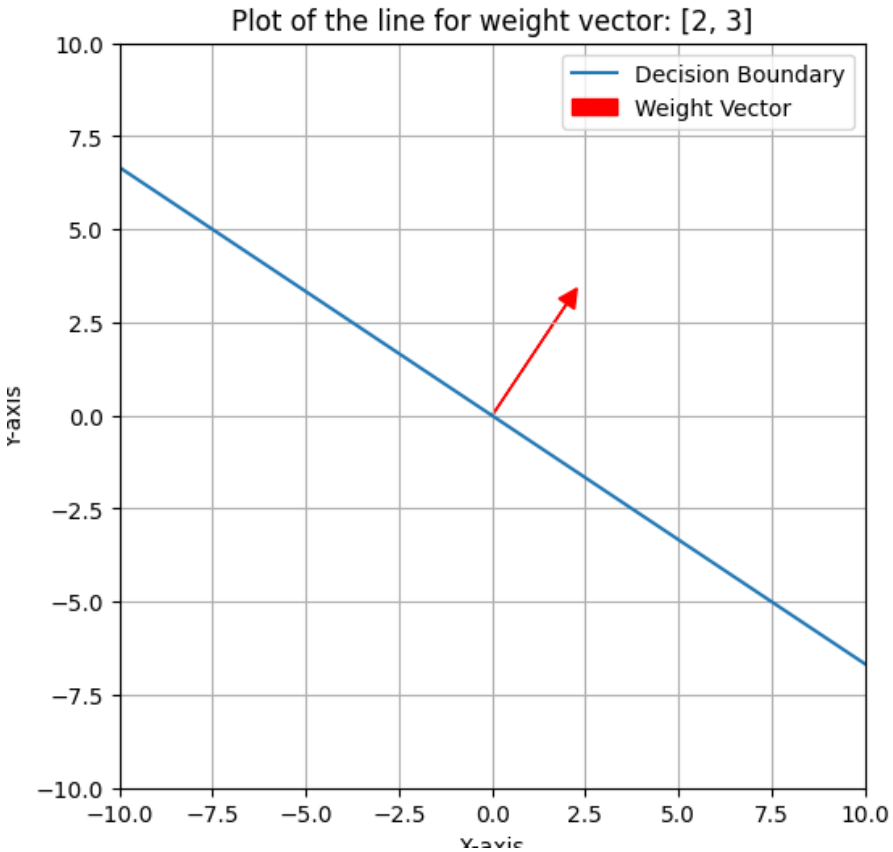
In a 2 dimensional (or really N-d) input data set, an SVM can be used to partition the data set using a “hyper plane”. For 2-d, this hyperplane becomes a line.

That line (also known as a decision boundary) can actually be represented by the equation:

```
w1*x + w2*y + w0 = 0
```

Where `[w1, w2]` is the weight vector and `w0` is the bias.

If `w0 = 0` (no bias) and `[w1, w2] = [2, 3]`; then it looks like this.



Note that the weight vector is itself the line. To draw the weight vector, start from the origin and make the vector (arrow) point towards `[w1, w2]`.

The weight vector is always perpendicular to the decision boundary as a rule.

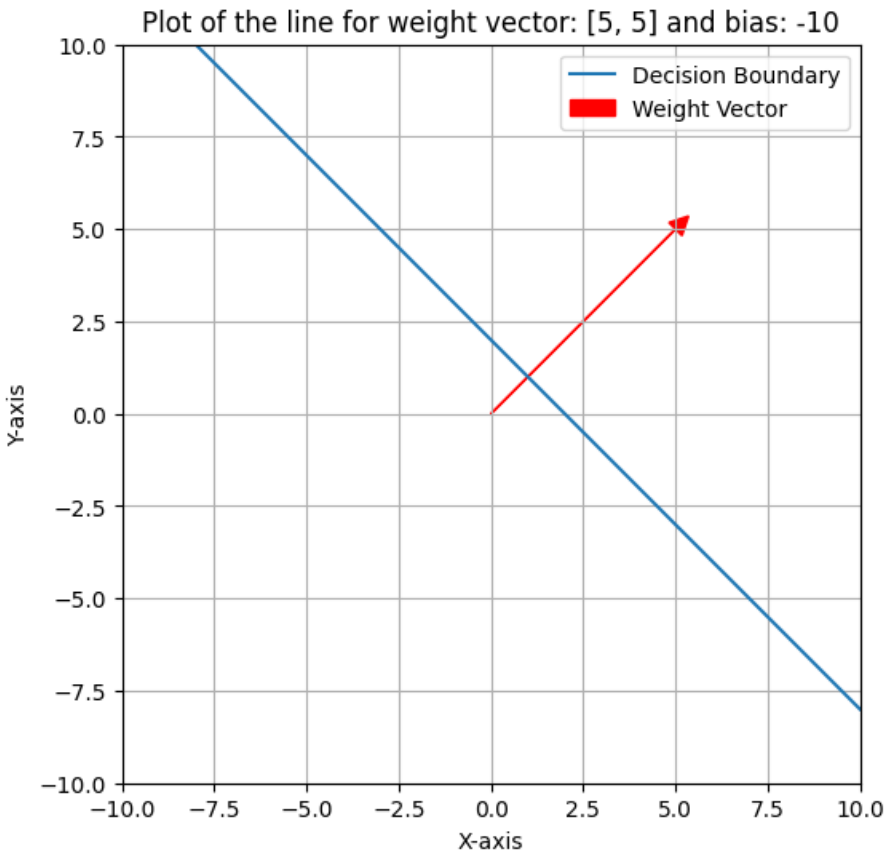
To draw the decision boundary, we can take the earlier equation for the decision boundary and transform it to the form `y = mx + c` to calculate the slope and the y intercept.

We get: `y = (-w1/w2) * x - w0/w2`

Plugging in the earlier weights, we get the blue line.

Now, so far `w0` has been 0. If we add some bias, the decision boundary shifts.

As an example:



As you can see, the decision boundary no longer passes through the origin, as some bias has been added. However, it is still perpendicular to the weight vector.

In conclusion, the weight vector and the bias together determine the decision-boundary line.

Mathematical partitioning of dataset using hyperplane / decision boundary

The earlier equation of the hyperplane in 2d space was a line: `w1*x + w2*y + w0 = 0`

Let us see how to intuitively check which side of this line does a random point (x, y) fall on.

If, for any point in the dataset (represented by x, y), `w1*x + w2*y + w0` is less than 0, then that point falls on one side of the line. If it is negative, it falls on the other side.

To see why, recall that by adjusting the bias `w0`, we were able to shift the decision boundary while still keeping it perpendicular to the weight vector.

Now consider an imaginary line passing through that random point (x, y). Assume that this line is perpendicular to the weight vector just like the decision boundary. This imaginary line can just be represented as a decision boundary with an adjusted bias. Let us see why.

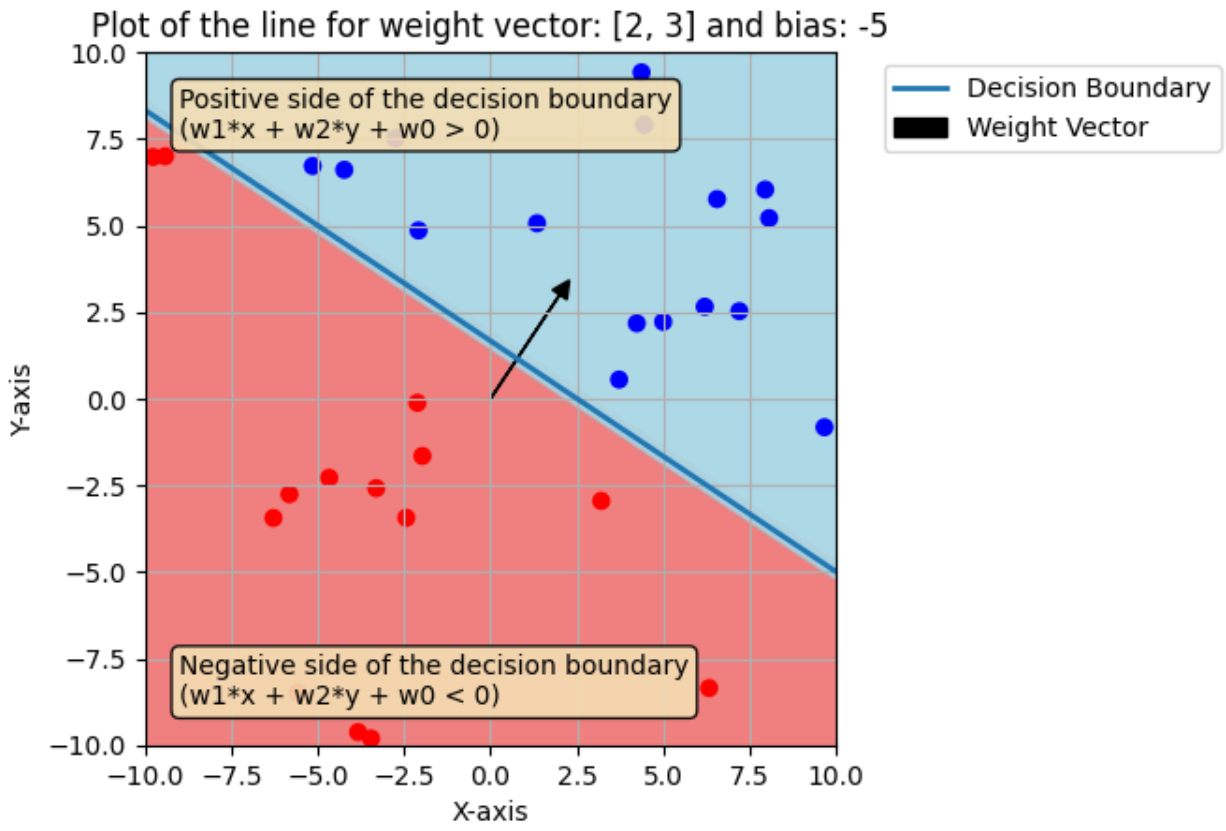
The equation of this imaginary line becomes `w1*x + w2*y + w0 - i0 = 0`, where `i0` is the adjustment to the bias for this line.

This becomes `w1*x + w2*y + w0 = i0`

From the earlier example with bias of `-10`, we know that if `i0` is negative, then it means such a line is parallel to the decision boundary and is shifted to the upper side. And if it is negative, it falls on the lower side. Thus, by just checking the polarity of `i0`, we can determine which side of the decision boundary it falls on. Obviously, if `i0` is 0, the imaginary line is the decision boundary itself.

To get `i0`, we simply plugin (x, y) into the decision boundary equation.

We can visually show this as below:



Compact hyperplane representation in N-dimensions

In an N-dimensional space, the hyperplane can be represented by increasing the dimensions of the weight vector, lets call this weight vector a capitalized W.

The hyperplane decision boundary can represented as the dot product.

In 2d, the hyperplane is a line as mentioned earlier:

```
w1*x + w2*y + w0 = 0
```

or

```
[w1, w2] . [x, y] + w0 = 0
```

or

```
Transpose([w1, w2]) * [x, y] + w0 = 0
```

In an N-dimensional space, consider the n dimensional input as `[x1, x2,… xn]` as vector `X`.

So we get the hyperplane decision boundary equation in N-dimensions as: `W . X + w0 = 0`.

As can be seen, `W . X` is 0 (give or take a bias). This is possible only if `W` is perpendicular to the decision boundary because the dot product of 2 vectors is 0 only if they're perpendicular to each other.

$\mathbf{A} \cdot \mathbf{B} = |\mathbf{A}|\,|\mathbf{B}| \cos(\theta)$

where $|\mathbf{A}|$ and $|\mathbf{B}|$ are the magnitudes of the vectors, and θ is the angle between them.

When the dot product is zero, it means that the cosine of the angle between the vectors is zero. The only way for the cosine of an angle to be zero is if the angle itself is 90 degrees.

SVM support vectors

Support vectors are the data points closest to the decision boundary in an SVM. They play a crucial role in determining the decision boundary and the margin. The margin is the distance between the decision boundary and the closest data points from both classes.

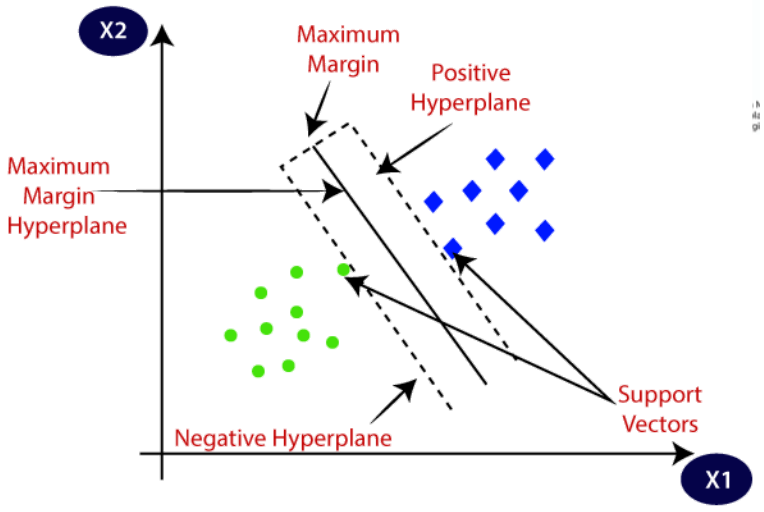
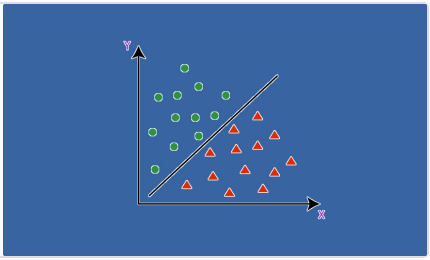


Diagram courtesy:

Support Vector Machine(SVM): A Complete guide for beginners

Support Vector Machine or SVM, is a powerful supervised algorithm that works best on smaller datasets but on complex ones.

https://www.google.com/url?sa=i&url=https://www.analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners/&psig=AOvVaw1pyCB_CC-HIjsmPPhIo68d&ust=16871955881



79000&source=images&cd=vfe&ved=0CBIQjhxqFwoTCLiQtaarzfECFQ
AAAAAdAAAAABAE

The "maximum margin hyperplane" is the decision boundary and the positive and negative hyperplane are called the support vectors.

After the SVM has been trained, the decision boundary lies at an equal distance from the positive and negative hyperplane and is a boundary that guarantees the maximum distance between the 2 support vectors. Any line that is partitioning the data but not result in a maximum distance is not an optimal decision boundary. A maximal distance ensures a more accurate and less erroneous model. The `W` vector and bias `b` are constantly adjusted till that is achieved.

Mathematically speaking, the positive and negative hyperplanes are basically the decision boundary adjusted with a bias. Because the decision boundary is at the exact distance from both these new hyperplanes, the bias is the same but just reversed.

So in the equation: `w1*x + w2*y + w0 = i0`, `i0` might be `5` for one hyperplane and `-5` for the other. But their absolute values will always be equal due to their decision-boundary-equidistance.

In fact to make the math easier, the 2 support vector hyperplanes can just be written as:

```
w1*x + w2*y + w0 = 1
```

```
w1*x + w2*y + w0 = -1
```

This is achieved by just dividing the weights and biases on the LHS of the equation by `i0`.

Such division as no effect on the decision boundary at all. Why?

Because, recall the decision boundary equation is:

```
w1*x + w2*y + w0 = 0
```

If we divide the LHS and RHS by `i0`, the result is still equal to 0 and the decision boundary does not move at all. The equation does not change. In fact, from earlier we know that translating this to the form `y = mx + c`, we know that the slope of this like is `-w1/w2` and the Y-intercept is `-w0/w2`. So the division of weights and biases by `i0` has no effect on the decision boundary line as these are just ratios.

Summarizing, in a 2d SVM system that is fully trained,

The decision boundary is represented as: `w1*x + w2*y + w0 = 0`

And the support vectors are:

Positive hyperplane: `w1*x + w2*y + w0 = 1`

Negative hyperplane: `w1*x + w2*y + w0 = -1`

From what we know about biases, it is easy to see that:

All the points outside the margin, on the positive side of hyperplane is: `w1*x + w2*y + w0 >= 1`

All the points outside the margin, on the negative side of hyperplane is: `w1*x + w2*y + w0 <= -1`

In these equations, we can represent `[w1, w2]` as the weight vector `W` and the point `[x, y]` as an input vector `X`. Let us represent the correct classification for a point as a scalar `Y` . And the bias `b` is `1 - w0`.

Plugging all this into the earlier equations, we can write a single equation for all points outside the margin as:

`Y * (W . X) + b >= 1`

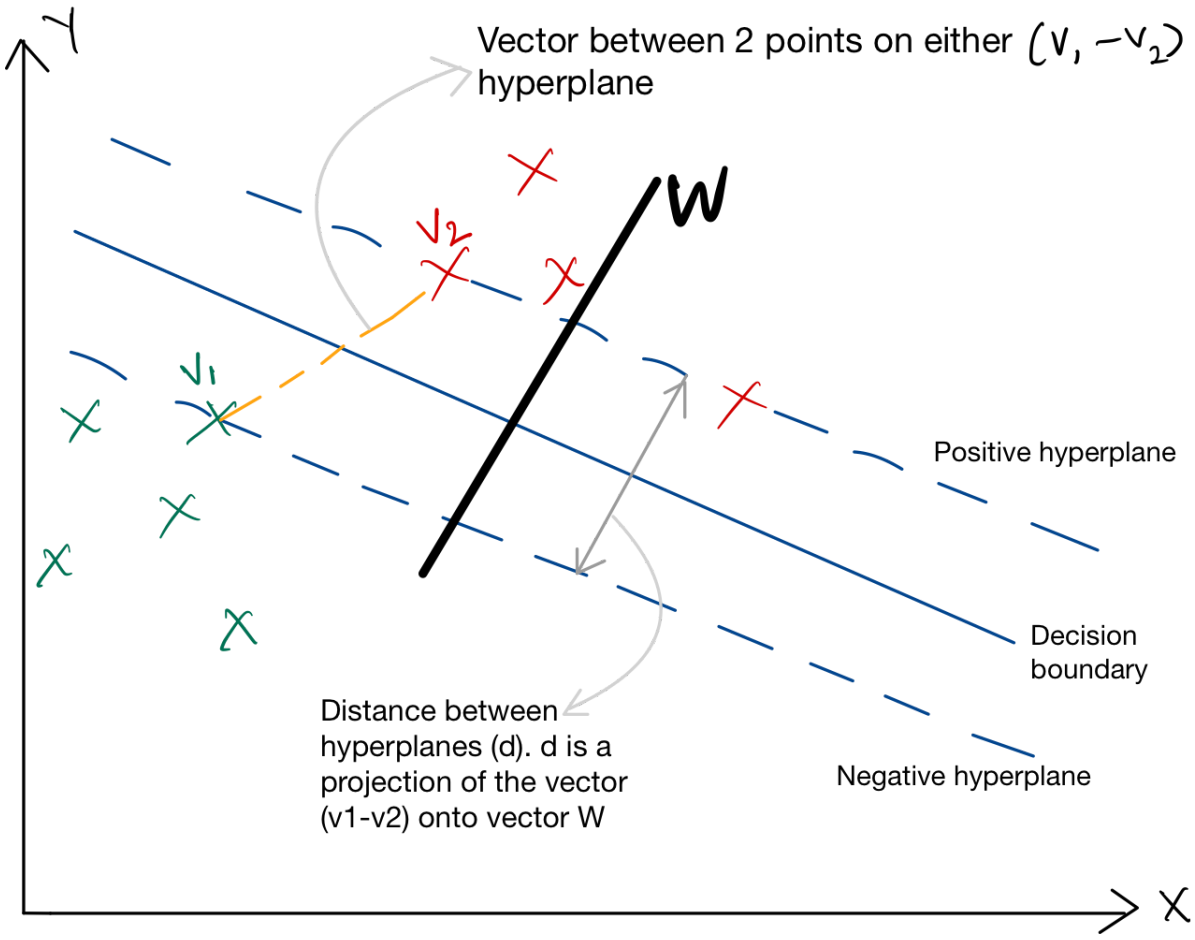
Maximizing of the distance between the support vectors

An optimal SVM draws its decision boundary such that the distance `d` between the positive and negative hyperplanes are maximum.

> 💡 Note that maximizing the distance can be done as long as the earlier equation from earlier is satisfied:
> This is also known as Hard SVM, because no points are allowed to lie in the margin. Later we will see how to relax that so we can widen `d` even more while tolerating somethings within the margin.

As we know, both the weight vector `W` and the bias `b` decide the position and orientation of the decision boundary.

The distance `d` constantly changes as decision boundaries are redrawn during training while `W` and `b` are adjusted. It is easy to see that 2 different decision boundaries might both linearly separate 2 classes of points, however the distance `d` between their support vectors may well be different. Depending on how the decision boundary is drawn, the support vectors might be closer or further apart from each other.



I made a mistake drawing `W` here, it should actually start at the origin. Further, it need not necessarily intersect the hyper planes.

In order to obtain the distance `d`, 2 points are selected on each support vector (hyperplane) and a new vector is drawn between them. In the figure, a new vector `V1-V2` is generated from the vectors `V1` and `V2` (orange line). See my other article on that concept. This new vector is then projected onto the weight vector `W` to find `d`.

For this, the dot product comes in handy! We can use the dot product between `W` and `V1-V2`` to assist in figuring out `d`.

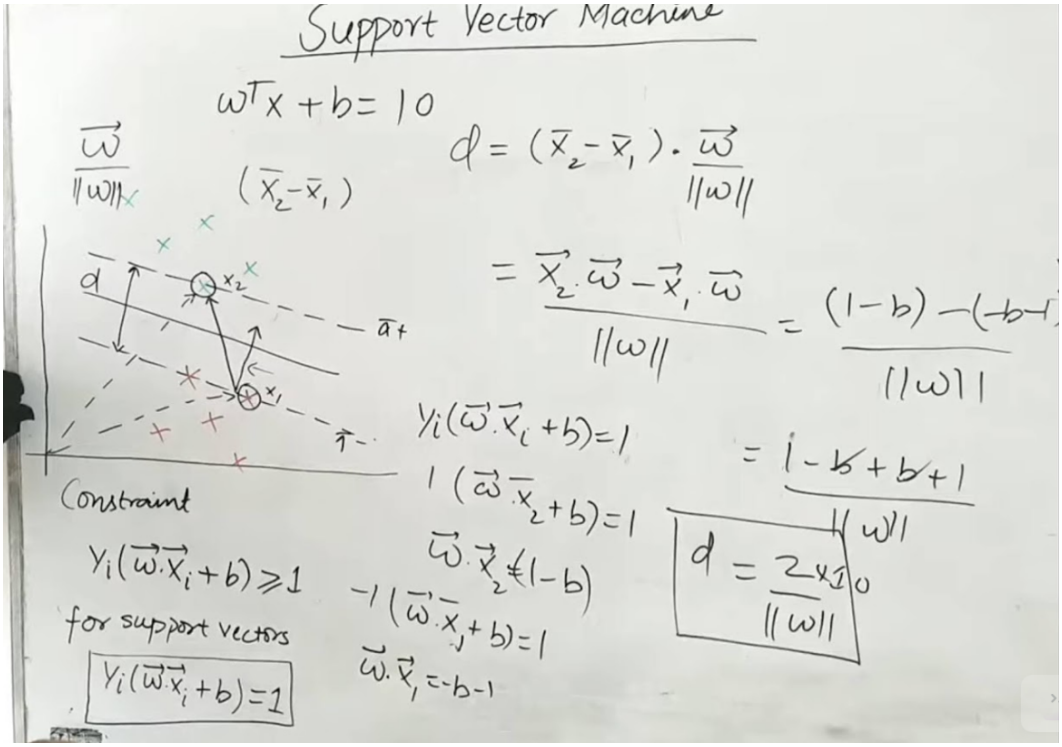
The formula for the projection of a vector `A` onto `B` is:

```
Magnitude of Proj_B(A) = |A.B| / |B|
```

See my article on Vector Basics for an explanation of that.

Thus, `d = ( (V1-V2).W ) / |W|`

This simplifies to `d = 2 / |W|`. Watch this youtube video or this image from that video for an explanation of the simplification:



Thus the goal to maximize `d` is to minimize `|W|`. However, as mentioned earlier, `d` can be increased as long as there are no points in the training data within the margin. In other words, maximization of `d` has to also keep the equation `` satisfied at all costs.

Allowing points within the margin

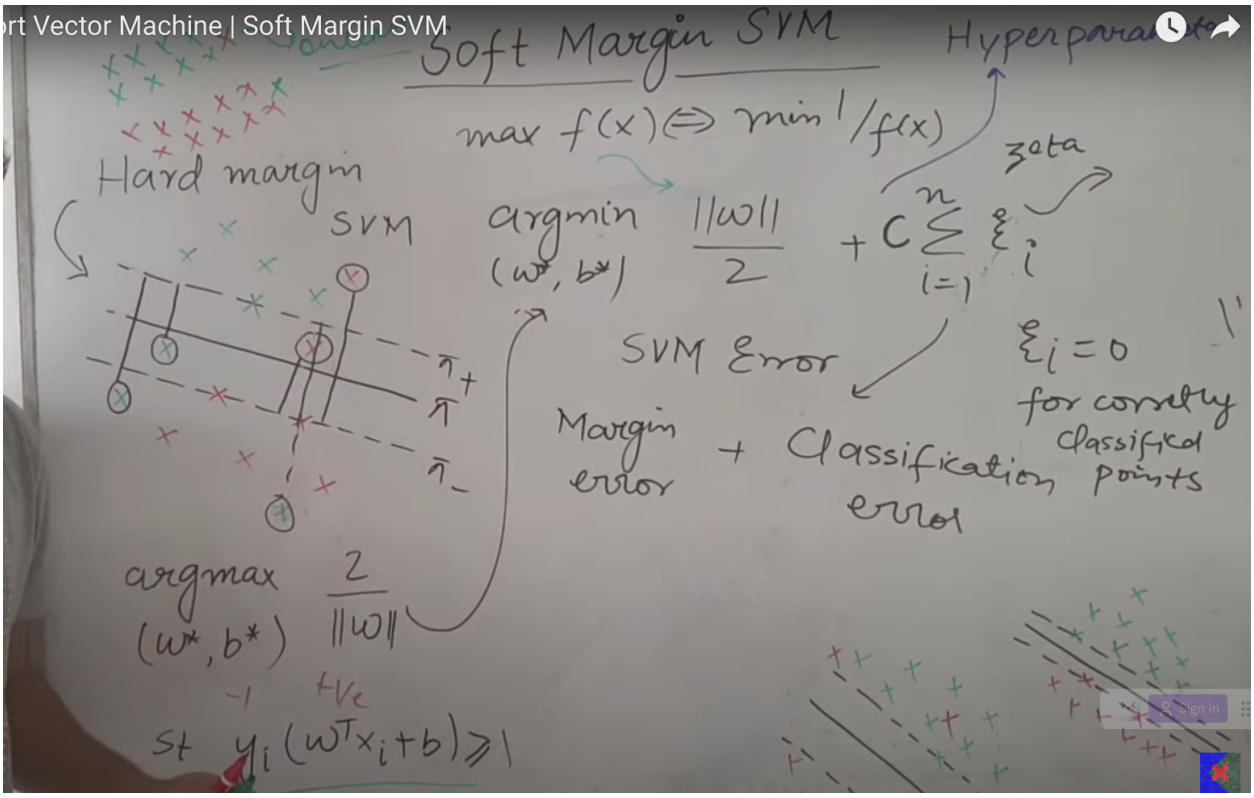
`d` can be maximized even more if we allow some training data to be ignored if they fall within the margin between the support vectors. Such points will have no influence of the weights `W` or bias `b` and thus no influence on the decision boundary's position or orientation. This is also known as soft SVM and lets the SVM tolerate more noise. It prevents over-fitting as well, as these noisy points within the margin have no influence on the decision boundary during training.

The following whiteboard shot from the above youtube video shows the softSVN equations.

1. The misclassified points are shown circled on either side of the decision boundary marked as pi. The support vectors are shows as pi+ and pi- and are between them

is the margin.

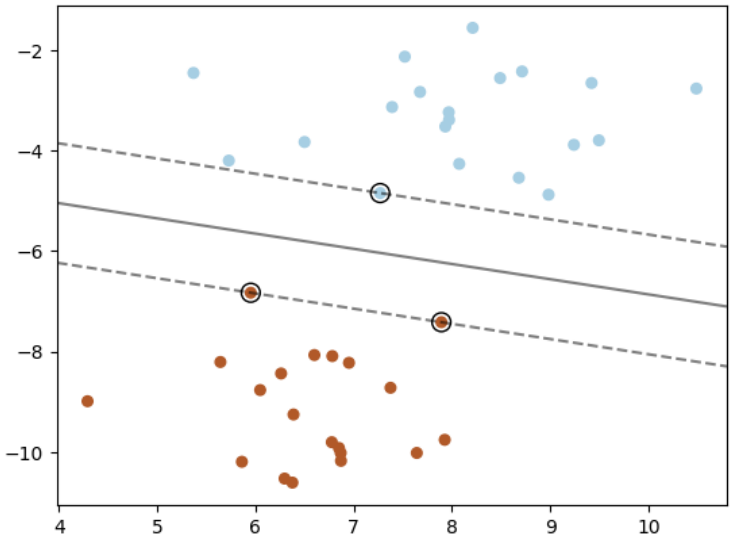
2. f(x) is the function (distance) we want to maximize. Such maximization can also be seen as “minimization” of margin error. After all, noise within the margin cannot alter a wider margin. `f(x) = 2 / |w|`.
3. Thus to reduce margin error, we want to minimize 1/f(x). That’s what the `argmin` term is in the whiteboard.
4. With soft SVM, there’s another term. Its called Zeta and is the “Classification error”. Each misclassified point adds a certain error. Together, these terms form the total SVM error or loss function. The goal during training is to minimize this loss.
5. The term `C` tells us how much does the classification error matter. A large value of `C` implies that misclassifications caus a lot of loss, the model tends to become a “harder” SVM.



Whiteboard screenshot from https://www.youtube.com/watch?v=yCAlHPDgWtM

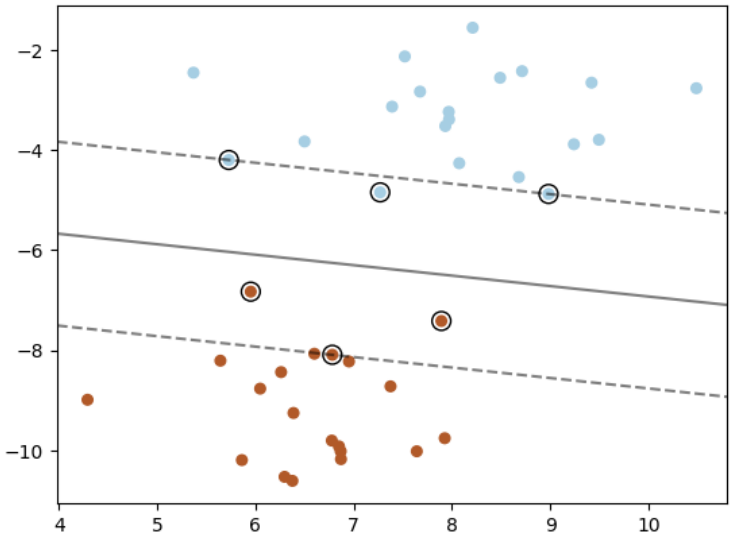
The following screenshot shows the effect of changing C on an SVM model generated by the scikit library.

With `C=1000`, we get a hard SVM:



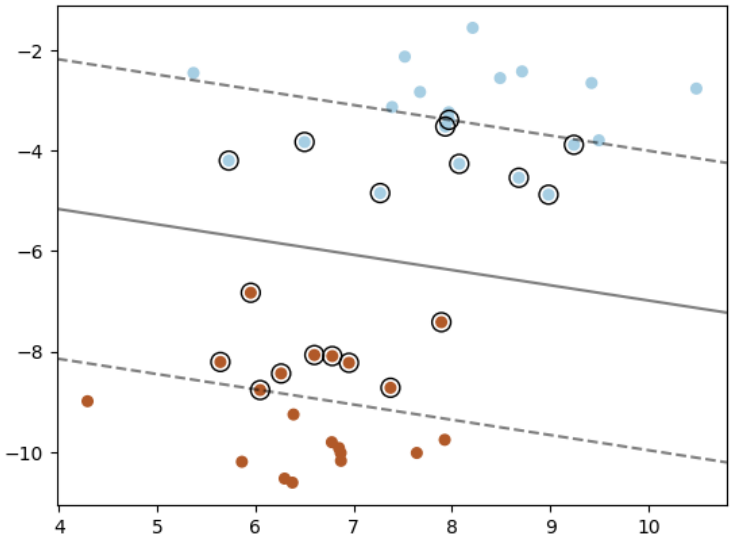
Hard SVM due to a large C. C=1000.

Reducing `C` to below 1 starts showing the SVM becoming softer. Here it is with `C=0.1`.

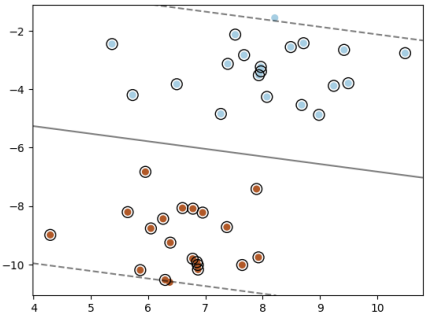


Soft SVM with C=0.1

Further reducing C shows more points added to within the margin. Here's `C=0.01`.



Soft SVM with C=0.01



SVM with C=0.02

Notice how the orientation (angle) of the decision boundary changes as the margin changes. This is because the points that were previously affecting the decision boundary are now within the decision boundary itself and are being ignored. With `C = 0.002`, we get an almost flat decision boundary:

Note that during inference, points falling within the margin are still classified depending on which side of the decision boundary those points fall under.

The `C` hyperparameter changes the importance of not having samples within the margin. Lowering `C` modifies the loss function, thus telling the training phase that it is more important to maximize the margin width `d` than to minimize the `zheta` errors. Thus the training phase ends up having more and more points within the margin.

Conclusion

This article just scratched the surface on getting an intuitive understanding of 2d SVM. There’s so much more to cover, such as kernels and so forth. However this article should lay out the basic intuitive knowledge to serve more advanced SVM concepts.