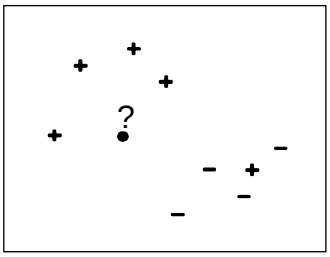
SVM – Friend or Foe?

We will go through what SVMs are and why they are good. SVMs are linear classifiers (a line in 2 dimensions, a plane in 3 dimensions, a n-1 dimensional hyperplane in n dimensions) and they are good for 3 good reasons and 1 very good reason.

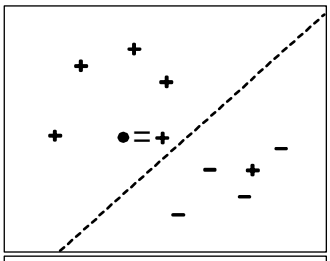
Reason 1

Consider the following labeled training data and a point that we want to classify.

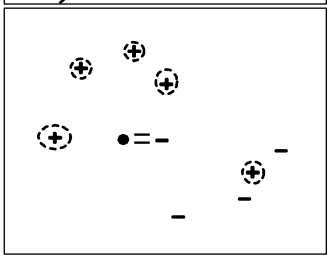


We classify the unknown point by fitting a classifier to the labeled training data (e.g. training) and then having it make the classification decision. Consider the following two classifiers:

This one is linear, everything on the left is classified as class **+**, everything on the right class **-**. It assigns class **+** to the point we are trying to classify.



This one essentially memorized class **+**. Everything in the immediate neighborhood of the **+** points is considered to be in the **+** class, everything outside is class **-**. It assigns class **-** to the point we are classifying.



Which one did better is a philosophical question. We will say that the linear one did a better job, you may agree with that statement, or you may disagree. There is no right answer here, but in the average case it appears like the linear classifier captured more from the data, we have higher confidence that if it sees more points from the same distribution it will label more of them correctly – that it will **generalize** better. This essentially is Occam's razor, all things being equal, we prefer simpler hypothesis; sometimes even if the simpler hypothesis does not correctly decide our training data – notice that the linear classifier misclassifies one of the **+** points.

A bit more formally the linear classifier is simpler since it needs less parameters. Any line

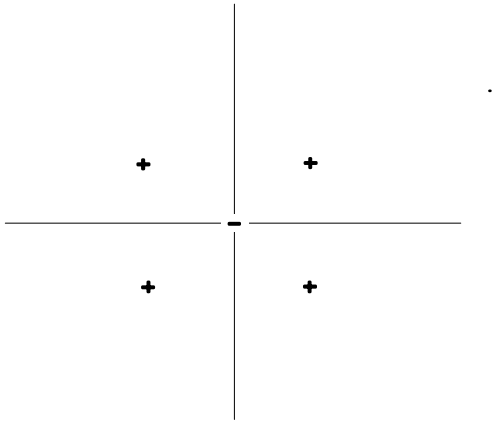
can be specified by two parameter m and b (as in y=mx+b), so the linear classifier is entirely determined by two parameters. The second classifier memorized 5 points, each with 2 coordinates, so it needs at least 10 parameters[1]. We say that the linear classifier has higher bias (=lower variance=lower capacity) than the second classifier, or equivalently that the second classifier has higher variance (=higher capacity= lower bias).

Take the above with a grain of salt, it as true as (and suffers from the same problems as) Occam's razor. Nevertheless in machine learning practice over-memorizing your data is a very common problem (called overtraining); not having your model have enough capacity does not happen nearly as much, so we'll go with it and say that:

LINEAR CLASSIFIERS ARE GOOD

Reason 2

Let us look at the following 2 dimensional dataset,



x	y	class
0	0	-
1	1	+
-1	1	+
1	-1	+
-1	-1	+

[1] To be honest, due to the author's laziness, as drawn the circles have different sizes, so the model was also able to specify a radius for each of the five circles – another 5 parameters for a total of 10+5=15. To be even more honest, due to even more laziness, some of the circles are really ellipses, so the model was able to fit 5 ellipses. The equation for an ellipse is $\frac{(x+r)^2}{a}+\frac{(y+q)^2}{b}=1$, so 4 parameters for each of the 5 ellipses; thus the model requires at least 4*5=20 parameters.

We have 5 points - 4 in class **+** and 1 in class **-**. This dataset is not linearly separable, we'll never be able to find a line (also called a **linear decision surface**, or a **linear classifier**) such that all the **+** are on one side and all the **-** are on the other. In order to try to make the data linearly separable we can compute more features from the data, for example we can add the feature x^2 to every data point. Our dataset represented in this **higher dimensional feature space** is then:

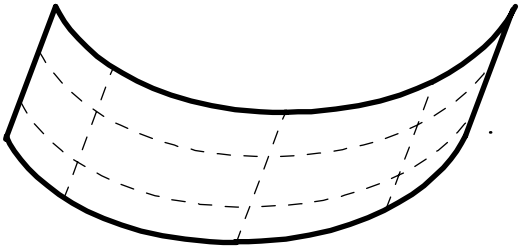
x	y	x^2	class
0	0	0	-
1	1	1	+
-1	1	1	+
1	-1	1	+
-1	-1	1	+

We see that

1) if $x^2=0$, the point is in class **-**
2) if $x^2=1$, the point is in class **+**

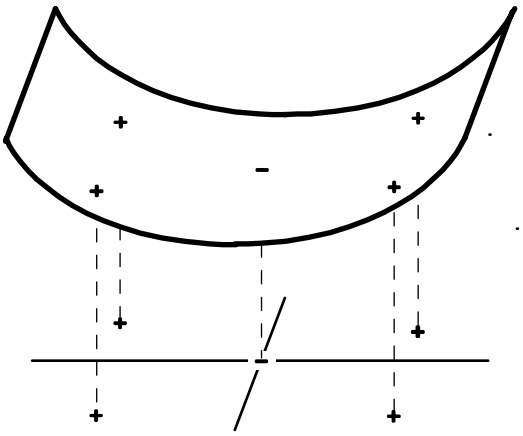
We have made the decision problem much simpler; let us consider what this looks like geometrically.

Here is the surface $f(x,y)=x^2$

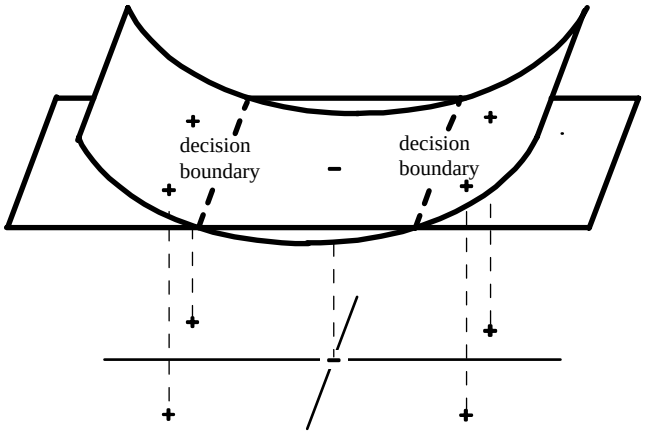


Let us plot our feature space points (x,y,x^2); they lie on this surface →

The **+** points lie above the **-** point, so the data is now linearly separable (a linear classifier in 3 dimensions is a plane).

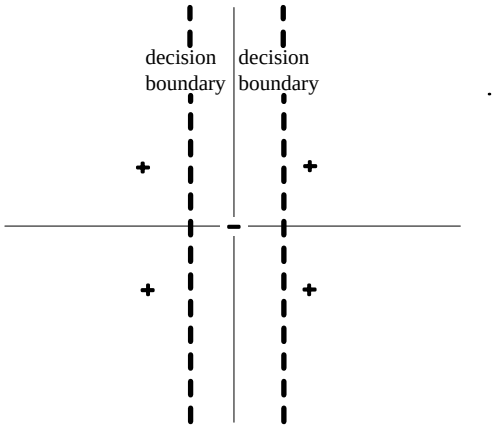


We can draw a plane such that **+** points lie above and the **-** point below. Our classifier (the plane) intersects the surface in two lines, we call this our **decision boundary** →



Class **-** is all points on the surface between these lines, class **+** is all the other points on the surface. What does this decision boundary look like in our original 2d space?

In feature space, the points are (x,y,x^2), to go back we drop the z-coordinate, which geometrically means to look straight down. Doing that we see that our decision boundary becomes →



Let us do another example, we will pick a different 3rd feature and see what happens. We select x^2+y^2 as our new feature. Proceeding as before we compute our new data.

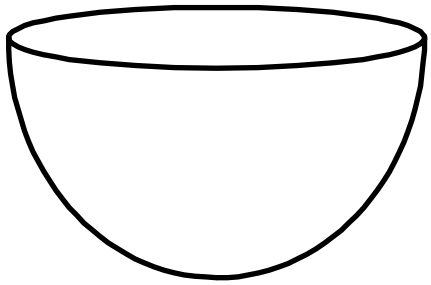
x	y	x^2+y^2	class
0	0	0	-
1	1	2	+
-1	1	2	+
1	-1	2	+
-1	-1	2	+

Again we notice

1) if $x^2+y^2=0$, then class **-**;
2) if $x^2+y^2=2$, then class **+**.

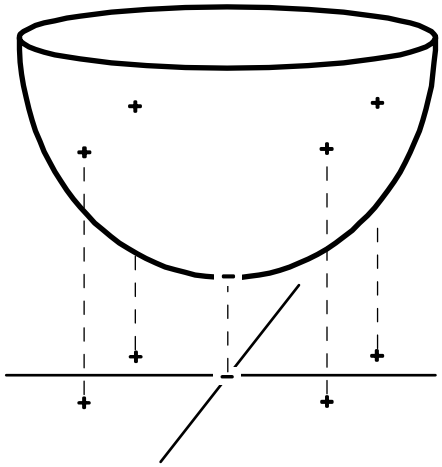
So we expect the data to have become linearly separable when considered in this higher dimensional feature space.

The surface $f(x,y)=x^2+y^2$



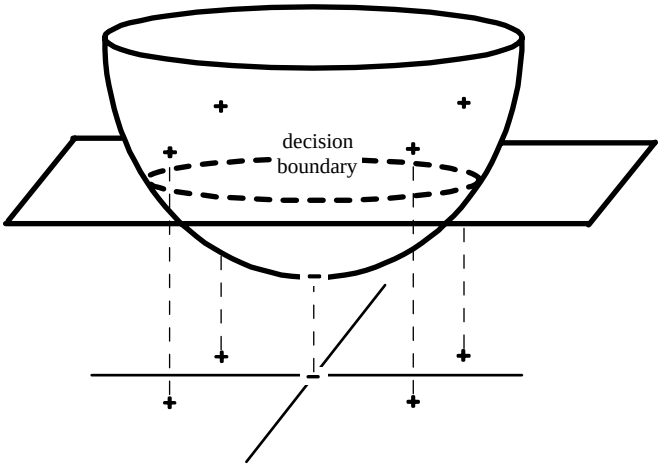
Here are our feature space points (x,y,x^2+y^2) →

They are indeed linearly separable though our original data is not.

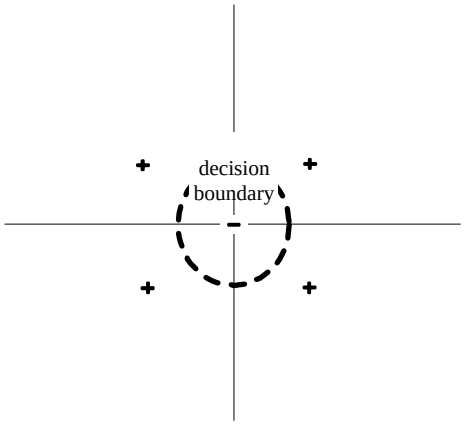


We find a plane (=linear classifier) such that class **+** is above and class **-** is below →

The intersection of the plane with the bowl-shaped surface is a circle; everything above that circle on the bowl is classified class **+**, everything below class **-**. Let us again consider what this decision boundary is in our original 2d space.



As before, we drop the z-coordinate by looking straight down and so the decision boundary back in our original space is a circle.



Thus we have the second reason why SVMs are terrific,

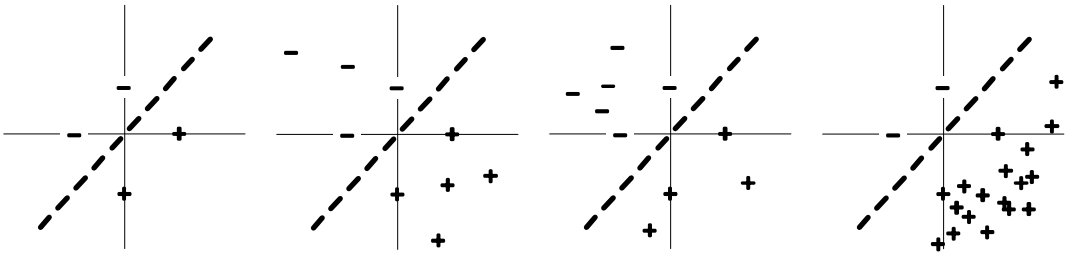
A HIGHER DIMESIONAL FEATURE SPACE IS GOOD

Because things that were not linearly separable before become linearly separable. Note that we never know which features will make our data linearly separable, x^2 and x^2+y^2 were lucky guesses, but if we pick many non-linear features (like xy, sin(x), $\log(x^2y)$, etc), then there is a good chance our data will become linearly separable, even though there is no certainty.

A caveat: an SVM does not just choose any plane that separates our data, it picks the one that maximizes the **geometric margin** between the data and the plane. In the example above, we could have picked a plane just a little bit below the **+** points – it would still separate the data, but we are better off picking the plane in the middle between the two classes. The task of finding this unique plane is an optimization problem – out of all possible planes we have to find the one that maximizes a certain constraint (the geometric margin). It turns out that this can be cast as a quadratic optimization problem, and we will not worry about it too much except to say that it can be done.

Reason 3

Consider the following datasets together with their optimal separating plane (which in 2 dimensions is a line)



We can see that only the four point that are alone in the leftmost picture matter in determining the separating plane. We call these points **support vectors**.

Let us concentrate on the case where we only have support vectors and let us consider how precisely we do the classification. The optimal linear classifier (in the geometric margin sense) that we see in the first picture on the left is the line y=x. This equation, written as 0=x-y means that any point on the line (like (3,3)) will give zero when we subtract y from x. However, the function f(x,y)=x-y can gives us more than that, for any point (x,y), if f(x,y)<0, then the point lies to the left of the plane; if f(x,y)>0, then the point lies to the right of the plane, so f(x,y) is the classifier. For any point $p=(p_x,p_y)$ that we wish to classify we decide p is in

1) class **+** if $f(p_x,p_y)>0$
2) class **-** if $f(p_x,p_y)<0$

We've seen that our classifier is completely determined by the support vectors, so it must be the case that the equation x-y is completely determined by our support vectors, but how? For each support vector we have three pieces of information, its two coordinates, and its classification, this has to be enough to get x-y. We have 4 support vectors: (-1,0) and (0,1) in class **-**; (0,-1) and (1,0) in class **+**. Let us use +1 for class **+** and –1 for class **-** . Then just summing together all the information from our support vectors we have

$$\textbf{-}\,(-1,0)\ \textbf{-}\,(0,1)\ \textbf{+}\,(0,-1)\ \textbf{+}\,(1,0) = -1*(-1,0)-1*(0,1)+1*(0,-1)+1*(1,0)=(2,-2)$$

and now its obvious that if we just dot product (x,y) with the result we get $(x,y)\cdot(2,-2)=2x-2y$. We are off by a constant, but the magic quadratic optimization procedure that find our optimal plane gives us a constant for each support vector, which in this case we see is ½, and we have our classifier, which in this particular example is

$$f(x,y)=(x,y)\cdot[-\tfrac{1}{2}\,(-1,0)-\tfrac{1}{2}\,(0,1)+\tfrac{1}{2}\,(0,-1)+\tfrac{1}{2}\,(1,0)]=x-y$$

To recap,

1) We start with our dataset
2) It uniquely determines a plane that best separates the data
3) We feed the data through a quadratic optimization procedure which finds this plane
4) The procedure tells us what this plane is by giving us a constant for every data point
    a. This constant is zero if the point is not a support vector (it must be zero, since as we saw before any point that is not a support vector can have no impact on the plane)
    b. Its non-zero for every support vector (in the example above this constant was ½ for every support vector, in general its going to be a different value for each)
    c. Caveat: if our separating plane does not pass through origin (as it did in the example above) the procedure will also give us a way to compute an intercept term as a function of the support vectors, but let us not worry too much about it here
5) To classify a point p
    a. If you are a person
        i. you look what side of the separating plane p is on
    b. If you are a computer
        i. you dot product p with every (support vector * its constant from the optimization procedure * its class value (+1 or –1)), and sum everything together. If the result is <0 then you decide p is in class **-**, if >0 p is in class **+** (if the result is 0, then p lies on the separating plane and you are screwed)

We have reason three:

THE CLASSIFICATION DECISION FOR A POINT *P* CAN BE EXPRESSED AS A FUNCTION OF DOT PRODUCTS OF *P* WITH THE SUPPORT VECTORS

If this reason seems a bit dubious, don't worry, it will come into its own in the next section.

Reason 4

As we saw above, classification of a new vector is mainly decided by taking dot products between our support vectors and the new vector we are classifying. Let us take a closer look at the dot product operation. Suppose our data is one dimensional, we have two vectors x and z; let us expand to a two dimensional feature space by adding a feature 3x. Then we have

$$\begin{array}{ccc} \text{original data} & & \text{data in feature space} \\ x & \xrightarrow{\text{expand}} & (x,3x) \\ z & & (z,3z) \end{array}$$

Let us take a dot product in feature space. We do it in the usual manner, multiply component by component and sum up,

$$\begin{array}{rcccc} (x,3x)\cdot(z,3z) = & & x & & 3x \\ & \times & z & & 3z \\ & & \overline{xz} & + & \overline{9xz} \quad = 10xz \end{array}$$

Now notice something **really important**: if we want to compute a dot product between two vectors in feature space, we can either expand each one and do the dot product in the usual manner, **or** we can just compute 10*xz*, which as we saw above is the same thing. This does not seem important only because the feature space expansion we've chosen here is trivial.

Consider another example, this time our data is two dimensional, we again have two vectors (x_1,x_2) and (z_1,z_2). We choose the expansion $(1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2)$. This is a very powerful set of features; remember in the examples before simply choosing x^2 as a feature made the data linearly separable, this particular expansion is more powerful, so we expect even more datasets that were not linearly separable in 2 dimensions to be linearly separable in this feature space. We have

$$\begin{array}{ccc} \text{original data} & & \text{data in feature space} \\ (x_1, x_2) & \xrightarrow{\text{expand}} & (1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2) \\ (z_1, z_2) & & (1, z_1^2, \sqrt{2}z_1z_2, z_2^2, \sqrt{2}z_1, \sqrt{2}z_2) \end{array}$$

Consider the dot product between two vectors in the feature space:

$$(1, x_1^2, \sqrt{2}x_1x_2, x_2^2, \sqrt{2}x_1, \sqrt{2}x_2)\cdot(1, z_1^2, \sqrt{2}z_1z_2, z_2^2, \sqrt{2}z_1, \sqrt{2}z_2) =$$

$$\begin{array}{cccccccccc} 1 & & x_1^2 & & \sqrt{2}x_1x_2 & & x_2^2 & & \sqrt{2}x_1 & \sqrt{2}x_2 \\ \times\ 1 & & z_1^2 & & \sqrt{2}z_1z_2 & & z_2^2 & & \sqrt{2}z_1 & \sqrt{2}z_2 \\ \hline 1 & + & x_1^2z_1^2 & + & 2x_1x_2z_1z_2 & + & x_2^2z_2^2 & + & 2x_1z_1 & + & 2x_2z_2 & = (1+x_1z_1+x_2z_2)^2 \end{array}$$

This means we can compute the dot product in this high dimensional feature space either the hard way - by expanding the vectors and doing the regular dot product, or the easy way - by computing $(1+x_1z_1+x_2z_2)^2$. Remember that to do the classification in the feature space all we need is to be able to take dot products. This means we can do the classification in this 6 dimensional feature space by simply computing $(1+x_1z_1+x_2z_2)^2$ from our original 2 dimensional data.

Now imagine our feature space is 1,000,000,000,000 dimensional – as long as the dot product in that huge space is the same as some simple function on our tiny dimensional, original data, we can extremely cheaply do linear classification in that enormous feature space.

We call $K((x_1,x_2),(z_1,z_2)) = (1+x_1z_1+x_2z_2)^2$ a **kernel function**. Every function that corresponds to the dot product in some particular feature space expansion is a kernel function, and given a kernel function we can do linear classification in that feature space. For example $K((x_1,x_2),(z_1,z_2)) = (1+x_1z_1+x_2z_2)^d$ for any $d>1$ is a kernel function, and is called a *polynomial kernel*[2].

Thus we have our final, very good reason why SVMs are good

WE CAN REPRESENT A DOT PRODUCT IN A HIGH DIMENSIONAL FEATURE SPACE AS A SIMPLE FUNCTION ON OUR ORIGINAL DATA

Which means we can classify in that feature space using this simple kernel function.

A recap: SVM classification is a name for the following

1) We take our data
2) We map it to a high dimensional feature space where it will probably become linearly separable
3) We find an optimal separating plane in the feature space
4) We are able to express the classification decision in the feature space in terms of dot products of vectors in the feature space

[2] Note that our original data can be n-dimensional; the polynomial kernel function is then $K((x_1,x_2,\ldots,x_n),(z_1,z_2,\ldots,z_n)) = (1+x_1z_1+x_2z_2+\ldots+x_nz_n)^d$.

5) We perform the classification decision by doing the dot products indirectly, using an equivalent kernel function which is much simpler, and much cheaper to compute