UNIT 2 : Non Linear Classification, Linear Regression and Colaborative Filtering

LECTURE 6 : Non Linear Classification

Wooooooooo !!!!!!! We are back baby !!!!! Yeahhhhhh !!!!!

After the successful completion of Fundamentals of Statistics , your pal **Scooby** is back with a brand new set of handwritten lecture notes. I found that many of you liked them and appriciated the work by thanking me personally on my email. Which by the way is :

Scooby's email : **peepeepoopoo873@gmail.com**

Feel free to point out mistakes and message me anyways

For those who do not message me...
I don't care man!!! Whataever man !!!! Screw you guys !!!! Whatever!!!

(I do care, please message me. I was just pretending to be indifferent in the above line so that you can't hurt me emotionally.) Anyways.

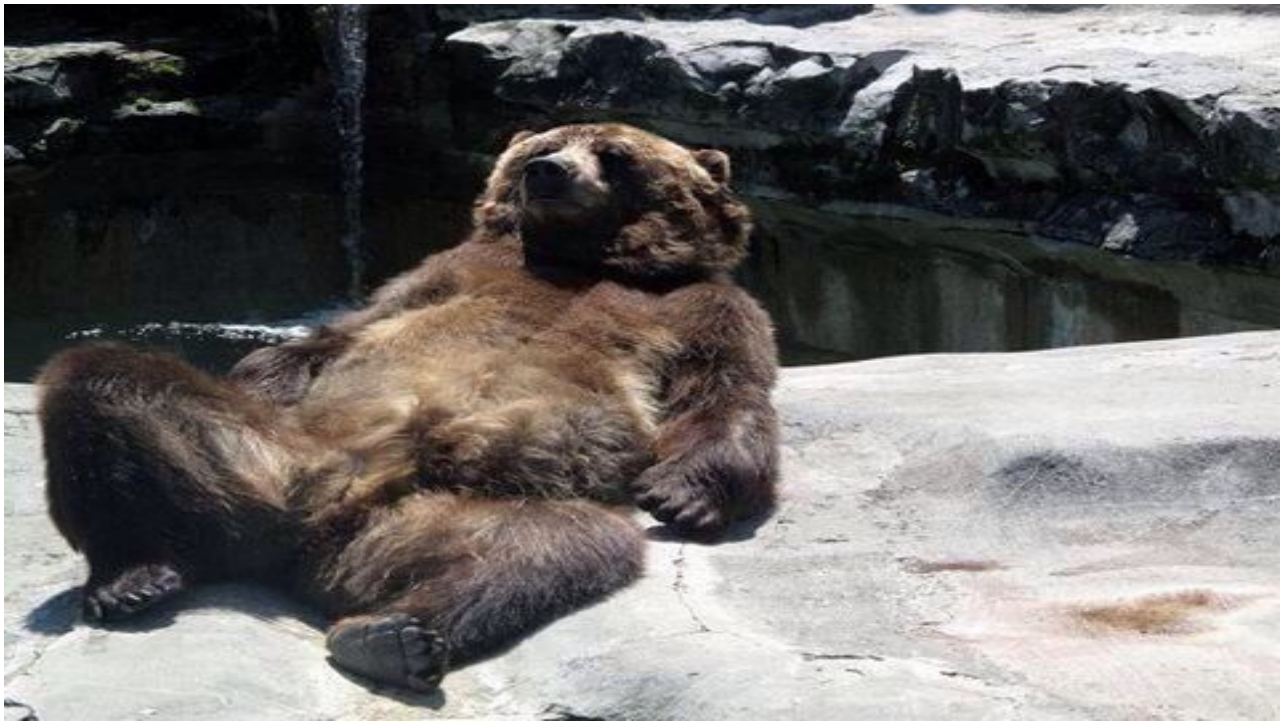
I tried to make these notes without any mistakes and cover most of the video material.

But as Yoda has rightly said : ' *Is human, To err* '

I however **DO NOT TAKE** any responsibilty if you fail your exams following my notes. So please check the actual lecture slides and videos if you suspect something is wrong, and mail me too.

The notes **DO NOT CONTAIN** solutions to any homework problems or in-lecture exercises. Only the problems and exercises solved in the videos are included in these notes.

Here is an interesting photo I googled



CHILLED BEAR

I ran out of made up quotes so I will write very short stories instead. These may or may not resemble true events, but will definitely have a message that will help you along the journey, that is life.

STORY TIME : The meaning of life

Once upon a time a man decided that he would figure out the meaning of life. He decided to travel the world, go places , meet people find the ultimate truth out. He travelled for years but could not find anyone who would guide him. He tried different religions and different idealogies but the answer was never there. Just as he was about to give up one day, he heard two women talking about a monk in the Shrine that was situated at a hill in that village.

The monk was well known in the village. There were rumours that the monk was able to heal people's pain. The villagers considered the monk as the incarnation of God itself.
The man thought, why not give it a try before quitting his quest. So he climbed the mountain top. It was high, and the weather was cold. But he didn't give up. He finally reached the shrine where the monk lived.

To his suprise, the monk was a chick (i.e, a female). He sat in front of her and told her everything about his journey and his quest. He finally asked her, what the meaning of all this was. Is there even an answer to that. Is that even the right question to ask.

“What is the meaning of life? " he cried
She stared at him for a while, then spoke ' I may have the answer to your question but it may not be the answer you want. In order to find your answer, you will have to do the work. I can only show you the way. '

So he pleaded for her to show him the way. She said that from tommorow on, he will have to sit in this place in the shrine and meditate on the meaning of life. In order to clear his mind out of all the crowd and distractions, he will first need to mellow his thought process. And what better that a marijuana loaded joint to achieve this task.

So it began, before every meditative session he smoked a joint and thought and thought for long streches of time. After six month or so, he took the monk's leave

“Did you find your answer?” asked Michelle (The monk's name was Michelle. Didn't expect that, did you?)
“No but I found out what to do.” replied the man “ I have to keep on searching the answer ”
“And how will you do that , after you leave this place?” asked Michelle

“ I will question things. I ask people's opinions on different things. I will interview all kind of crazy people with some crazy opinions. I will ask different questions, and seek different answers. We will talk about aliens, simulations and technology and psychology and evrything else. Maybe I will record these interviews. There's this new site called Youtube, I will upload my videos on that. Maybe some people will be iterested in listening to all this. A buddy of mine knows all this stuff, he will help me. I wonder where Jamie is these days. I will have to discover my body too, so maybe I will pick up a sport to play or something. And the most important thing of all” said Joe with a spark in his eyes “ I will tell the world that cannabis should be legalised !!!!!!”

THE END
MORAL : Cannabis is a great tool to explore the meaning of life

I hope these notes will help you

Let's get started

▷ Linear Classifier on the real line

Lets start with a very simple problem to see how ideas evolve. We have a real valued x, that is $x \in \mathbb{R}$ and we want to explore the classification problem again.

The setup is such that we have two points

labelled $+1$ @ $x = 1$

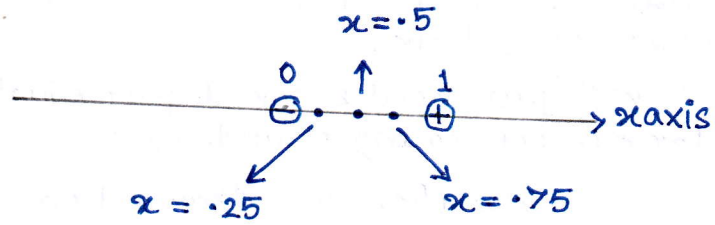
labelled -1 @ $x = 0$

& if we want to classify these points or make a classifier using this train set then we can use the help of our hyperplane in this setup

Generally we used $\theta \cdot x + \theta_0$ to classify the points.

Hyperplane as discussed earlier is an $n-1$ dimensional thingy in a n dimensional space. Here we have a 1 dimensional space as $x \in \mathbb{R}^1$ so our hyperplane will be $\theta x + \theta_0 = 0$, a 0 dimensional thing, which is a point on the x axis

For eg $x = \cdot 5$; $x = \cdot 75$; $x = \cdot 25$ are some of the possible hyperplanes (points) that correctly classify the data points above.



This is shown in the adjoining figure.

We can then make our classifier $h(x)$ by taking the sign of the expression $\theta x + \theta_0$ for any general x

$$h(x) = \text{Sign}(\theta x + \theta_0)$$

For eg. If we decide $x = \cdot 5$ is the hyperplane (point) to be used we can write it as $x - \cdot 5 = 0$ and make the classifier as

$$h(x) = \text{Sign}(x - \cdot 5)$$

which means points to the right of $\cdot 5$ will be labelled $+1$
points to the left of $\cdot 5$ will be labelled -1

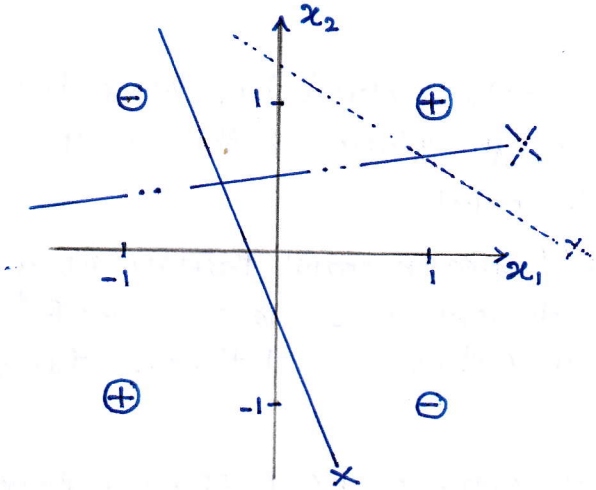
This was possible because the data we have (the two points) is linearly seperable. If that was not the case, we could have never found a hyperplane able to classify all points correctly.

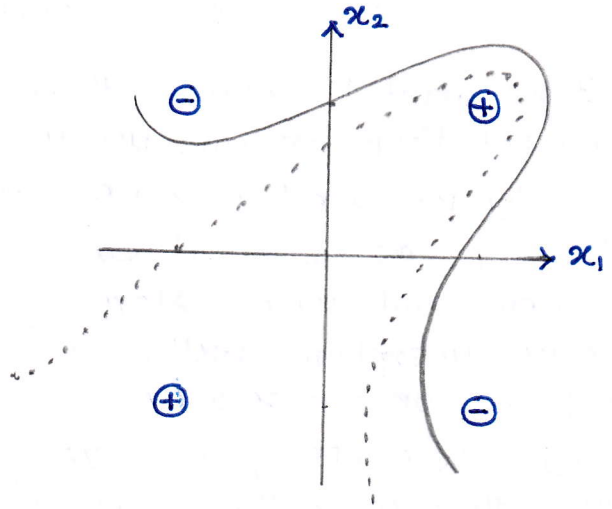
But this does not mean we can't seperate these points at all. We just can't linearly seperate them.

For example. A very simple case we saw was when $x \in \mathbb{R}^2$ & we had points

$+1$ @ $(1,1)$
-1 @ $(-1,1)$
$+1$ @ $(-1,1)$
-1 @ $(1,-1)$

The picture looked like below





There is no linear classifier possible for this train set because no line (hyperplane) can classify all points correctly. This is because data is not linearly seperable.

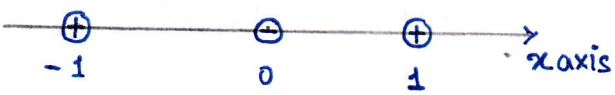
Three classifier's planes that are linear are drawn above

But this does not mean that there are no classifiers that can properly classify the points, all of them.

We will just need a non linear equation for our seperating boundary

Two such classifiers are drawn here

We will come back to this example again, but lets first go back to the previous 1d case of points on the real line ie $x \in \mathbb{R}$



If we add another point labelled -1 @ $x = -1$ we see that the problem becomes non seperable (linearly)

There is no point (hyperplane) that would be able to classify all points correctly.

These things motivates us to expand our arsenal of method and incorporate some new tools to tackle these problems.

But linearity was a pretty cool thing too and it had a very neat mathematical structure so we also want to retain that. Let's see how we proceed

➤ Feature Transformation

The idea we will use will be pretty simple. We will make a new feature vector from the old feature vector

The old feature will be as usual denoted by x. In the present example that we are studying x is a real valued scalar, but the formalism is being generated for all cases.

The new feature vector will be denoted by : $\phi(x)$

The idea behind $\phi(x)$ is that it is a vector function of x

we can say $\phi(x)$ looks like

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_m(x) \end{bmatrix}$$

where each

$$\phi_i(x) : \mathbb{R}^d \to \mathbb{R}$$

is a real valued function that takes in the old feature vector $x \in \mathbb{R}^d$ (generally) & gives a real value $\phi_i(x)$

How many entries or functions we want the new feature vector $\phi(x)$ to be made of will depend on the complexity of our problem and our computational power, and we will address that issues later.

But first lets get the idea clearly

- → We have the old feature vector $x \in \mathbb{R}^d$ generally
- → And we are making a new vector from the information in x
- → This new vector denoted by $\phi(x)$ will usually be a vector in dimensions more than d
- → Each element / component for this new vector $\phi(x)$ will be a multidimensional real valued function $\phi_i(x)$ that will take as input the old feature vector x, and output a real value.

This way we are transforming our feature vector x

Now an important thing that we want is to be able to go from our feature vector $\phi(x)$ to the old feature vector x. Because we don't want any original information (collected data) to get lost in the translation

As an example suppose $x \in \mathbb{R}$ and we propose the following $\phi(x)$

$$\phi(x) = \begin{bmatrix} x^2 \\ x^4 \end{bmatrix}$$

Now suppose my algorithm predict's a buy / sell for a stock & it says Buy when $\phi(x) = \begin{bmatrix} 4 \\ 16 \end{bmatrix}$. x was the data collected and suppose x denotes the % return of the previous day of the stock

So I collected x : % return of previous days for different stocks.
In order to simplify the problem I applied $\phi(x)$ to each old vector x to form new feature vector $\phi(x)$, and told my algorithm to make predictions & fit models with this new feature vector $\phi(x)$.

Now my computer says

Buy when $\phi(x) = \begin{bmatrix} 4 \\ 16 \end{bmatrix}$ so I would want to know what this means in terms of x : % return of previous day

So I solve for x by : $\begin{bmatrix} x^2 \\ x^4 \end{bmatrix} = \begin{bmatrix} 4 \\ 16 \end{bmatrix}$ which will give me $x = +2$ OR -2

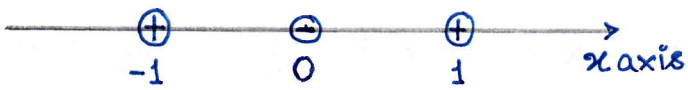
Now I am confused. Should I Buy when return in +2% or −2%
This happened because while going from x to $\phi(x)$ we lost information

So when ever we are making a new feature vector $\phi(x)$ we will make sure that the first d components of $\phi(x)$ are the same d components of vector $x \in \mathbb{R}^d$. Ie

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \vdots \\ \phi_d(x) \\ \phi_{d+1}(x) \\ \phi_{d+2}(x) \\ \vdots \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \\ \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

where we have $x \in \mathbb{R}^d$ $\quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{bmatrix}$

With that being said, let's revisit our example earlier



we had three points
+1 @ $x = -1$
−1 @ $x = 0$
+1 @ $x = 1$
& we want a classifier that can properly classify them all.

Suppose we suggest the following feature transformation

$$\phi(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$$

See how the 1's component is just x

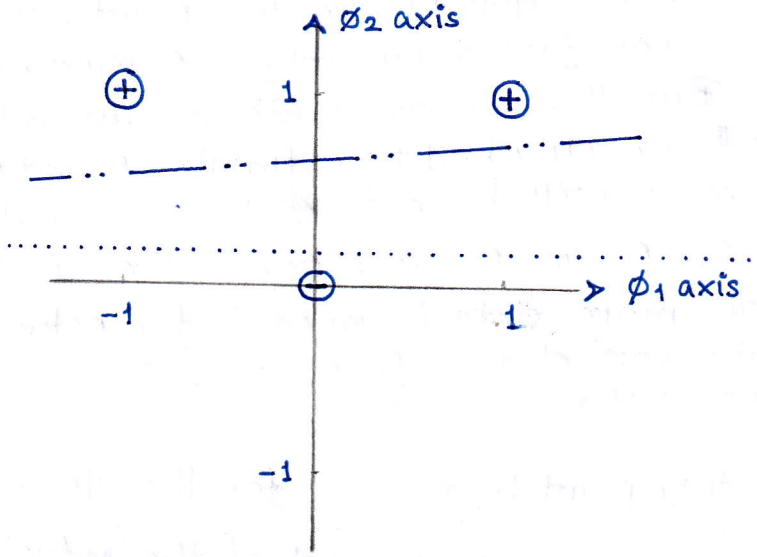
Let's see how our points transform

$+1$ @ $x = -1 \implies$ @ $\phi(-1) = \begin{bmatrix} -1 \\ 1^2 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

-1 @ $x = 0 \implies$ @ $\phi(0) = \begin{bmatrix} 0 \\ 0^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$+1$ @ $x = 1 \implies$ @ $\phi(1) = \begin{bmatrix} 1 \\ 1^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Let's see/visualize how these examples look in the new feature space



Aaahaaaa !!!

This is just great. The points now are clearly seperable in the new feature space. In fact I have drawn two (of many) boundries that can do this job for us and these are linear boundaries

Now we can write the equation of hyperplane in this new feature space too Because we are in a 2d space $\theta = [\theta_1 \ \theta_2]^T$ and the input vector is $\phi(x) = [\phi_1(x) \ \phi_2(x)]^T$. The equation looks like

$$\theta \cdot \phi(x) + \theta_0$$

ie $\quad \theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_0 = 0$

If we plug in $\phi_1(x) = x$; $\phi_2(x) = x^2$ we finally have

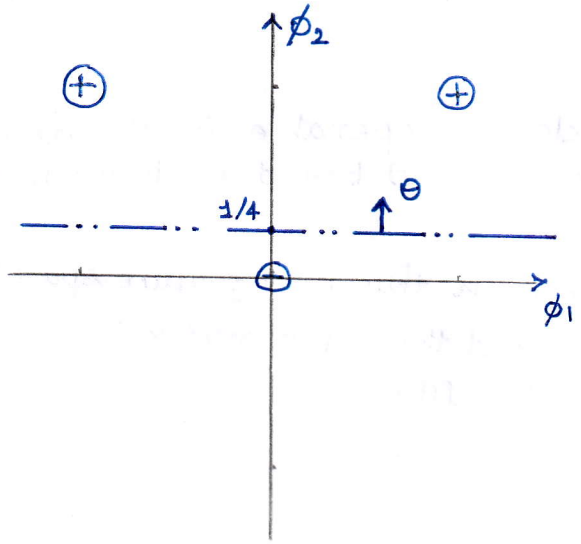
$$\theta_1 x + \theta_2 x^2 + \theta_0 = 0$$

Thats pretty interesting, and we will see why

> This is interesting because, the data that was not seperable in 1 dimension is clearly linearly seperable in two dimensions. Using this trick of forming a new higher dimensional feature vector $\phi(x)$ from the old Lower dimensional feature vector x, is pretty cool. Because it allows us to formulate the problem's solution as a hyperplane of higher dimensions. But it still is a hyperplane, so all the algorithms like perceptron, SGD etc will still be applicable.

Look at it this way. The case we had just now for $x \in \mathbb{R}$ had no Linear classifier that can classify all points correctly. But we used the trick of making a 2 dimensional vector $\phi(x)$ from the old vector x and found out that, in this new picture the points were infact Linearly seperable & we can find a hyperplane $\theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_0$ that would do the job. But that would come at an additional cost. Earlier when $x \in \mathbb{R}$ we only had to estimate θ and θ_0 for our analysis. But here we will have to estimate one additional θ ie we need to estimate θ_1, θ_2 and θ_0 ie 3 parameters instead of 2 More parameters mean more data is needed for estimation. And we will have to take care of this fact while increasing the dimensions for feature vectors

> There's another interesting point here and for that let me take a specific example



Suppose out of the infinite choices I have for a classifier. I choose the hyperplane

$$\phi_2 = 1/4$$

as shown in the figure

In it's general form we can write

$$\phi_2 - 1/4 = 0$$

Here $\theta_1 = 0$; $\theta_2 = 1$; $\theta_0 = 1/4$

Similiarly we can have the classifer

$$h(x; \theta, \theta_0) = \text{Sign}(\theta \cdot \phi(x) + \theta_0)$$

In our case $\equiv \text{Sign}(\phi_2 - 1/4)$

Although I wrote ϕ_2 we must not forget that its actually $\phi_2(x) = x^2$

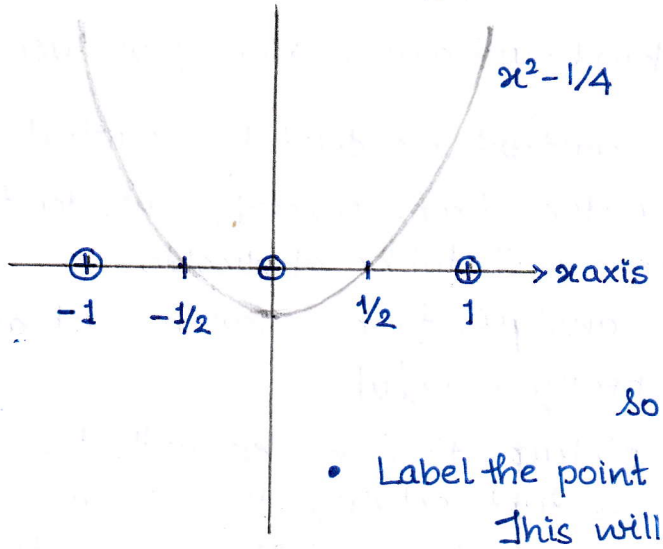
I wanna see how this picture is playing out in the 1 dimensional (original feature space)

So we have the classifier $h(x;\theta;\theta_0) = \text{Sign}(\phi_2 - 1/4)$

We also know $\phi_2(x) = x^2$ So the classifier actually is

$$h(x,\theta,\theta_0) = \text{Sign}(x^2 - 1/4)$$

Lets plot this thingy : $x^2 - 1/4$ to see whats happening



We have also put our labelled points at their respective positions

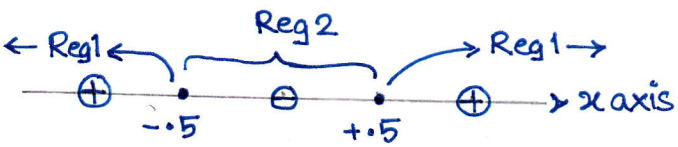
$x^2 - 1/4$ is actually the equation of a parabola that cuts the x-axis at $1/2$ and $-1/2$

ie $x^2 - 1/4 = 0$ @ $x = \pm 1/2$

So the classification rule is actually saying

- Label the point $+1$ when $\text{Sign}(x^2 - 1/4)$ is positive
  This will happen when x is either $> 1/2$
  OR x is either $< -1/2$
- Label the point -1 when $\text{Sign}(x^2 - 1/4)$ is negative
  This will happen when $x^2 - 1/4 < 0$
  ie $x \in (-1/2, 1/2)$

So if we look only at the 1d picture



The x axis is still being divided into two region but this time two points (instead of 1) are dividing it

Reg 1 : $(-\infty\ -1/2) \cup (1/2, \infty)$

Reg 2 : $(-1/2, 1/2)$

The interesting thing happening here is this.

Suppose for a minute we did not know anything about this feature vector transformation method. And we still want to classify the points previously

we clearly see that a single point can't classify all the train set properly.

ie $\theta_1 x + \theta_0$ will be of no help here

Suppose now someone comes to us and say why don't you use a non linear classifier?

What is that we ask?

He says, instead of using the equation of line $\theta_1 x + \theta_0 = 0$

Use the equation of parabola $\theta_1 x + \theta_2 x^2 + \theta_0 = 0$
This new equation has term containing x^2. Hence its a non linear equation. He tells us that this will be able to classify all points correctly

But the problem is more complicated now

> How do we find a new objective? Earlier we had the interpretation of $\frac{(\theta \cdot x + \theta_0)y}{\|\theta\|}$ being the signed distance. We penalized our points accordingly, $\frac{1}{\|\theta\|}$ was the distance bw boundaries. Now if we use equation like $\theta_1 x + \theta_2 x^2 + \theta_0$ in this context we don't know what it means. We will have to form our objective from scratch. We don't have any guarantee that the perceptron will still be applicable. In total we will have to do the whole analysis from scratch just as we change the equation & That's pretty painful.

But if we know the transformation picture. than we know that using the function $\theta_1 x + \theta_2 x^2 + \theta_0$ is just solving the linear problem in 2 dimension instead of 1. So I have all my intuition intact. And I can apply all my algorithms to this case just by telling my computer to make a small change. Just take the input data x and transform it to $\phi(x)$ for all cases in training data.
Now perform the same linear analysis as earlier on this new $\phi(x)$

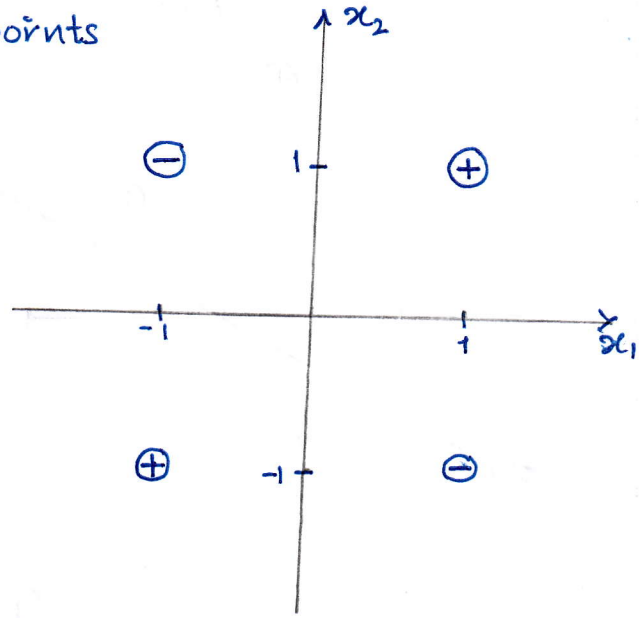
▷

"Can you think about how that would look ? you sexy beast....."

▷ Let's see another example

we have seen this problem earlier. Four points

Label	Position
+1	(1, 1)
−1	(−1, 1)
+1	(−1, −1)
−1	(1, −1)



Now this we know is a non linearly seperable set, so there's no point in trying to fit a hyperplane (line here) in this region.

But suppose we suggest the following transformation

for $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$ we do

$$\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \in \mathbb{R}^3$$

As discussed earlier we put the first two components of $\phi(x)$ to be the compo-nents of x vector itself and the third component $\phi_3(x) = x_1 x_2$

So

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \\ \phi_3(x) \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_1 x_2 \end{bmatrix} \in \mathbb{R}^3$$

Let's see how the points transform

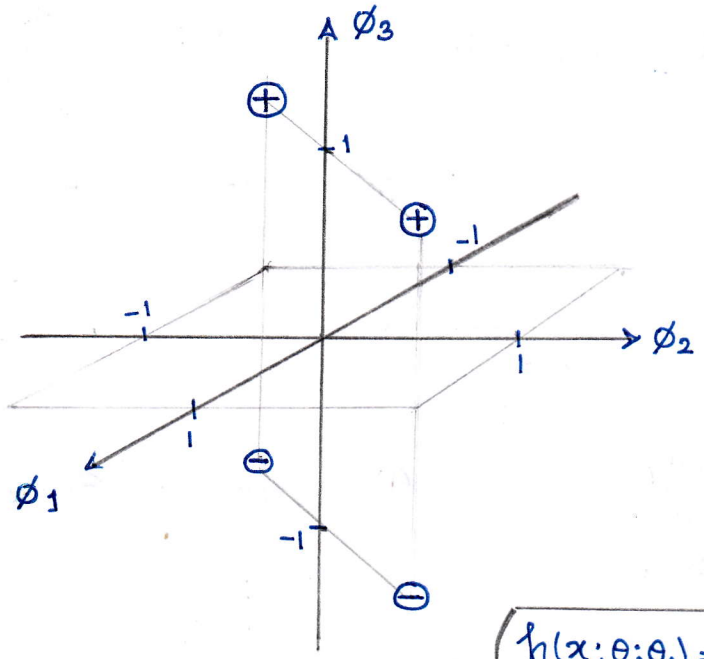
$$+1 \quad @ \quad x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \implies \phi(x) = \begin{bmatrix} 1 \\ 1 \\ 1 \cdot 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$-1 \quad @ \quad x = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \implies \phi(x) = \begin{bmatrix} -1 \\ 1 \\ -1 \cdot 1 \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

$$+1 \quad @ \quad x = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \implies \phi(x) = \begin{bmatrix} -1 \\ -1 \\ -1 \cdot -1 \end{bmatrix} = \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix}$$

$$-1 \quad @ \quad x = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \implies \phi(x) = \begin{bmatrix} 1 \\ -1 \\ 1 \cdot -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ -1 \end{bmatrix}$$

Let's see how the picture evolves in this new feature space



Aaahaa!!!

Success again. We clearly see that the new points are seperable in this new feature vector space

We can make our classifier using the hyperplane

$$\theta \cdot \phi(x) + \theta_0 = 0$$

$$\theta_1 \phi_1(x) + \theta_2 \phi_2(x) + \theta_3 \phi_3(x) + \theta_0 = 0$$

& the classifier would be

$$\boxed{h(x;\theta;\theta_0) = \text{Sign}\,(\theta \cdot \phi(x) + \theta_0)}$$

Let's see what the picture in our original feature space.
For that let's again see a specific example.
We will take the example for a specific plane in $\phi_1 - \phi_2 - \phi_3$ space
we clearly see that a plane perpendicular to ϕ_3 axis passing through origin classifies all the points correctly. So let's study that

- The θ vector will be along ϕ_3 axis: $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$ would do our job because θ vector is perpendicular to the plane we choose

The plane will be $\begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \end{bmatrix} + \theta_0 = 0$

Le $\phi_3 + \theta_0 = 0$

We also note that the plane passes through $(\phi_1, \phi_2, \phi_3) = (0,0,0)$

So $0 + \theta_0 = 0 \implies \theta_0 = 0$

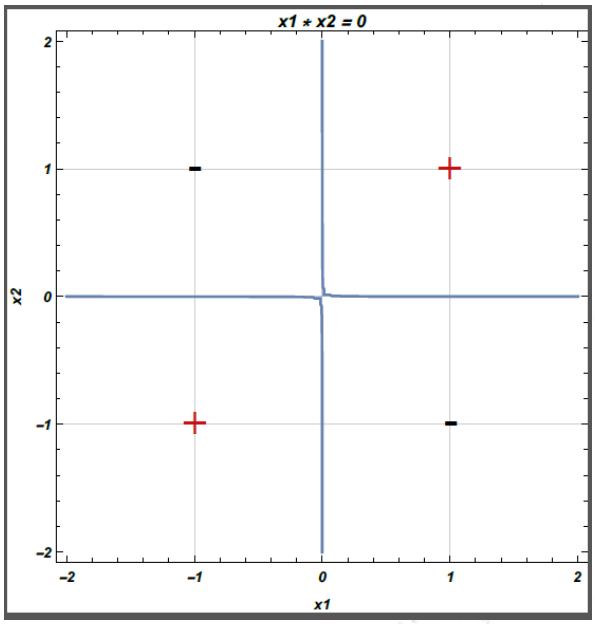
So the equation of plane is $\phi_3 = 0$

In order to go to $x_1 - x_2$ plane we substitute $\phi_3(x) = x_1 x_2$ as decided earlier

What does this equation represent

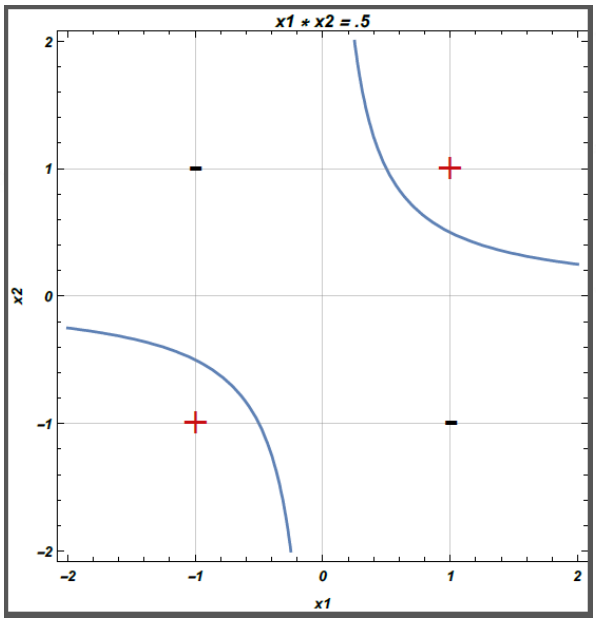
$x_1 x_2 = \text{constant}$

represents the equation of rectangular hyperbola



This is how the equation $x_1 x_2 = 0$ looks when draw on x_1-x_2 plane

see how the curves are dividing the region in into two parts and the decision boundaries are not linear



If suppose I took the plane perpendicular to ϕ_3 axis but, it passed through say $(0, 0, +.5)$ then the equation of plane will be

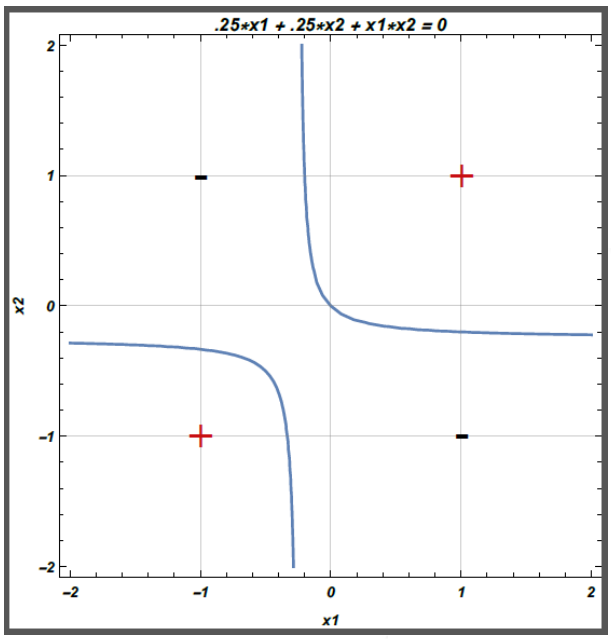
$$\phi_3 - .5 = 0$$

this is equivalent to

$$x_1 x_2 - .5 = 0$$

$$x_1 x_2 = .5$$

which looks like this (see adjoining figure)



Let's say now I tilt the plane a little so that it still classifies all points correctly and passes through origin.

Suppose the θ vector is $[1/4, 1/4, 1]$

The equation of plane is

$$\frac{1}{4}\phi_1 + \frac{1}{4}\phi_2 + \phi_3 = 0$$

The equation becomes

$$\frac{x_1}{4} + \frac{x_2}{4} + x_1 x_2 = 0$$

which looks like this (see adjoining figure)

The point is we can now classify the points using non-linear boundaries with our new formalism.

▷ Polynomial Features

Once we have established this idea that a non linear classifier in the original feature space can be seen as a linear problem in higher dimensional transformed feature space $\phi(x)$ we can add more and more dimensions to make our life simpler

Suppose $x \in \mathbb{R}$ is our original input/feature vector

We can write $\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \\ \vdots \\ x^p \end{bmatrix}_{p \times 1}$ & do our analysis on this feature space.

Lets see some cases for clarity

- Lets say $x \in \mathbb{R}^2$ ie it's two dimensional

$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$ and suppose we make a feature vector transform containing 2nd order features which look like

$\phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \end{bmatrix}$ Now this new feature vector lives in 5 dimensions

$\phi(x) \in \mathbb{R}^5$

Now using this feature vector I can classify my ith point

$h(x; \theta, \theta_0) = \text{Sign}(\theta \cdot \phi(x) + \theta_0)$

OR I can even do regression if labels are continuous real values

$f(x; \theta, \theta_0) = \theta \cdot \phi(x) + \theta_0$

- Let's look at the example when $x \in \mathbb{R}$

I can with the help of this new formalista take

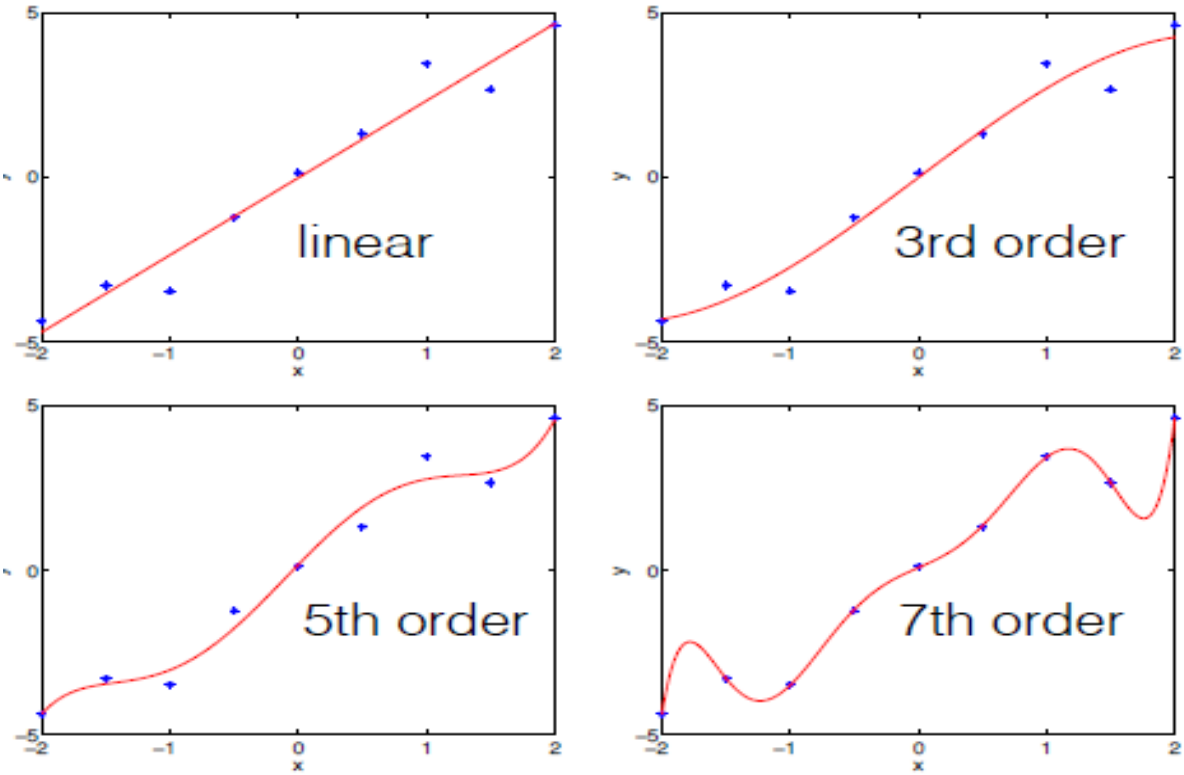
$\phi(x) = x$: linear case

$\phi(x) = [x, x^2, x^3]^T$: 3rd order polynomial

$\phi(x) = [x, x^2, x^3, x^4, x^5]^T$: 5th order polynomial

$\phi(x) = [x, x^2, x^3, x^4, x^5, x^6, x^7]^T$: 7th order polynomial

Prof T. plots these fits for us, for a given train data set he has



But as model complexity increases, a problem start to emerge. We see what the problem is. We will again run into the problem of our model overfitting the data. And we won't be able to generalise our model to the test data.

We already know the solution to this. We can perform cross validation to find the optimal solution.

Properly Doing cross validation, Prof T. find out that the linear case is the optimal case.

But this is not the only problem we face.

Lets look at the case for $x \in \mathbb{R}^d$

The feature vector most generally will have

1st order features $[x_1, x_2, x_3 \ldots\ldots x_d]^T$

2nd order features $[x_1^2, x_2^2, x_3^2, \ldots\ldots x_d^2, x_1x_2, x_1x_3 \ldots\ldots]$

So a total of $d \times d = d^2$ features

3rd order features $[x_1^3, x_2^3 \ldots\ldots x_d^3, x_1^2x_2, x_1^2x_3 \ldots\ldots x_1x_2x_3 \ldots]$

So a total of $d \times d \times d = d^3$ features

& so on

If d is moderately small like 100, the feature vector still becomes very high dimensional very quickly.

If we plan to transform $\phi(x)$ with all p order terms we will have $d + d^2 + d^3 + \cdots\cdots + d^p$ terms in $\phi(x)$ if we plan to use a pth order polynomial transformation.
we will have many many terms that will increase the complexity very very fast

For $d = 100$ we will have

$$100 + 10000 + 1000000 + 100000000 + \cdots\cdots$$

So when we try to apply an algorithm like say perceptron this new ϕ-space that this algorithm will work on, will have feature vectors $\phi(x)$ of billions or maybe trillions of dimensions, and the task will be extremely computationally intensive (maybe even impossible)

But due to the beauty and convinience of this feature vector transform picture, we also want to use these methods.

So in order to tackle this dilemma we will have to be mathematically cunning, and we will do just that. For us to solve this problem the first tool we will need is.

▷ Kernels

The idea of introducing kernels is in the line of us being hopeful, that the computational complexity will decrease. So as we go through this part let's keep tab of the steps / calculations performed by the computer while performing the steps
Lets start with a simple example

1) Suppose we have an input vector $x \in \mathbb{R}$, ie a 1 dimensional input
Suppose we have the simplest feature transform

$$\phi(x) = \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix}$$

A polynomial of degree = 2

NOTE
The 1 in the 1st entry can be kept to find θ_0 like $\theta \cdot \phi(x)$ will be like $\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix}$

There is also another vector that we denote by x'
x' will also be similiarly transformed

$$\phi(x') = \begin{bmatrix} 1 \\ \sqrt{2}x' \\ x'^2 \end{bmatrix}$$

For some reason we want to take the inner product of these two transformed vectors

$$\phi(x) \cdot \phi(x') = \begin{bmatrix} 1 & \sqrt{2}x & x^2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}x' \\ x'^2 \end{bmatrix} = 1 + 2xx' + (xx')^2 = (1 + xx')^2$$

The way we described the transform with the 1 & $\sqrt{2}$ gives us

This neat form

Lets see how many steps the computer will take to compute

$$\theta \cdot \phi(x)$$

We want to check the steps for this computation, because we plan to apply the perceptron algorithm to our feature space, And if you recall there was this step

Check if $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$ for the ith point

But this was the case when we talked about finding a hyperplane in x vectors space. Here our new vector are the transformed versions of the x vectors ie $\phi(x)$. And we are trying to find a hyperplane in this new feature space. So the check will be

Check if $y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$

Therefore we have to see how much computationally complex is

$$\theta \cdot \phi(x)$$

Lets see we had

$$\phi(x) = \begin{bmatrix} 1 \\ \sqrt{2}x \\ x^2 \end{bmatrix} \text{ and } \theta \text{ will be like } \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

So to find $\theta \cdot \phi(x)$ = 3 mutiplications of each element, 3 additions (actually 2) > ≈ 6 steps

We somewhy also want to find the computation complexity of calculating

Yeah! that's a word

$$\phi(x) \cdot \phi(x') = (1 + xx')^2$$

→ 1 step to find xx' Recall $x, x' \in \mathbb{R}$
→ 1 step to add 1
→ 1 step to square

3 steps

3 steps or 6 steps, what's the big difference, but we move on.

Lets see, let's make the problem a little interesting.

12> We still say that $x \in \mathbb{R}$ and $x' \in \mathbb{R}$

But now we are interested in a 4th order polynomial transform

Let's see

$$\phi(x) = \begin{bmatrix} 1 \\ 2x \\ \sqrt{6}\,x^2 \\ 2x^3 \\ x^4 \end{bmatrix} \quad \text{And } \phi(x') = \begin{bmatrix} 1 \\ 2x' \\ \sqrt{6}\,x'^2 \\ 2x'^3 \\ x'^4 \end{bmatrix}$$

Don't worry about these $2, \sqrt{6}, 2^{2/3}$ these are just for our ease.

Lets see what $\phi(x)\cdot\phi(x')$ is

$$\phi(x)\cdot\phi(x') = 1 + 4xx' + 6(xx')^2 + 4(xx')^3 + (xx')^4$$
$$= (1 + xx')^4$$

Lets again see how many steps

$\theta\cdot\phi(x)$ takes ? $\longrightarrow$ 5 steps for dot product
5 steps for adding
(actually 4)
$\approx$ 10 steps

While

$\phi(x)\cdot\phi(x')$ takes
$= (1+xx')^4$
$\longrightarrow$ 1 step for xx'
1 step to add 1
1 step to power 4
$=$ 3 steps

|3> Lets see the general case when $x, x' \in \mathbb{R}$

$\phi(x)$ will be a p dimensional polynomial transform

Its elements will look like $1, x, x^2, x^3 \ldots\ldots\ldots x^p$

To calculate $\theta\cdot\phi(x)$ $\longrightarrow$ p+1 steps for dot product
p steps for addition
$>$ 2p steps

To calculate $\phi(x)\cdot\phi(x')$ we will need 3 steps

because $\phi(x)\cdot\phi(x')$ will also be like : $(1+xx')^p$

So although the steps needed to check $\theta\cdot\phi(x)$ keeps on increasing as p increases, the steps to calculate $\phi(x)\cdot\phi(x')$ still remains constant.

Lets see what happens when we go to 2 dimensions

14> Let me show you how this works out when we have

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2 \quad ; \quad x' = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix} \in \mathbb{R}^2$$

Suppose we transform x as a 2nd order polynomial

$$\phi(x) = [1, x_1, x_2, x_1^2, \sqrt{2}\, x_1 x_2, x_2^2]^T$$
$$\phi(x') = [1, x_1', x_2', x_1'^2, \sqrt{2}\, x_1' x_2', x_2'^2]^T$$

$$\phi(x) \cdot \phi(x') = \underbrace{1}_{\substack{\text{this is} \\ \text{a number} \\ \text{called one}}} + \underbrace{x_1 x_1' + x_2 x_2'}_{\substack{\text{this is} \\ x \cdot x'}} + \underbrace{(x_1 x_1')^2 + 2(x_1 x_1')(x_2 x_2') + (x_2 x_2')^2}_{\substack{\text{this is} \\ (x \cdot x')^2}}$$

$$= 1 + x \cdot x' + (x \cdot x')^2$$

If I had been a little careful and would not have directly copied the example from the slides, and had put the appropriate real coefficients (like before $\sqrt{2}, \sqrt{6}$, etc) before terms like $x_1, x_2, x_1^2 \ldots$ I could have written:

$\phi(x) \cdot \phi(x')$ as $(1 + x \cdot x')^2$

My point is even when x is 2 dimensional. We can find a polynomial transform of order p that will gives us

$$\phi(x) \cdot \phi(x') = (1 + x \cdot x')^p$$

And lets now finally compute the steps needed for the most general case

15> We take the most general case when $x \in \mathbb{R}^d$ & $x' \in \mathbb{R}^d$ and $\phi(x)$ will transform x into a polynomial of order p

Basically $\phi(x)$ will have terms like

$x_1, x_2, x_3 \ldots \ldots \ldots x_d$: d terms

$x_1^2, x_2^2, \ldots x_1 x_2, x_1 x_3 \ldots x_d x_1 \ldots$: d^2 terms

$x_1^3, x_2^3, \ldots x_d^3, x_1^2 x_2, x_1 x_2^2 \ldots$: d^3 terms

$\vdots$

$x_1^p, x_2^p, \ldots x_d^p, \ldots x_1^{p-1} x_2, \ldots x_1 x_2^{p-1} \ldots$: d^p terms

So now we can see how much steps will be needed to

Check $\theta \cdot \phi(x)$

A total of $d + d^2 + d^3 + \cdots\cdots d^p$ multiplications of the dot product
And d additions of the dot product

So basically d^p is the dominant term here because d is usually large, think like 10000, for large scale ML and the other terms like d^{p-1} d^{p-2} ....d are small comparable to d^p

Lets see how much computing $\phi(x) \cdot \phi(x')$ costs

$\phi(x) \cdot \phi(x')$ will still be $(1 + x \cdot x')^p$

$x \cdot x'$ will take d multiplications
1 step to add 1
1 step to take to power p

} Aprox d steps need.

So its a matter of d^p steps vs d steps

d^p steps → to calculate $\theta \cdot \phi(x)$

d steps → to calculate $\phi(x) \cdot \phi(x')$

But now we have had all this discussion let me first make some comments
This function that we made by dotting two transformed vectors is called a Kernel function

$$K(x, x') : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$$

$$K(x, x') = \phi(x) \cdot \phi(x')$$

It is a function that takes in two dimensional vectors and gives a real value $\mathbb{R}$
We will study about kernels a little later. But first
We must ask ourselves. Why are we even comparing $\theta \cdot x$ with $\phi(x) \cdot \phi(x')$
What if one's complexity is very high d^p, while the others is very low d. It's not
Like we can use $\phi(x) \cdot \phi(x')$ as a check in our perceptron instead of $\theta \cdot \phi(x)$

OR CAN WE ???

In the next section we will see how the perceptron algorithm can be applied to a higher dimensional feature space, but the computational complexity will still remain under control. and this will be due to this new thingy called Kernel we defined here

Let's do it...

▷ Kernel Perceptron

Let me first remind you how the perceptron algorithm worked or looked

procedure PERCEPTRON($\{(x^{(i)}, y^{(i)}), i = 1, \ldots, n\}$, T)
  $\theta = 0$ (vector)
  for $t = 1, \ldots, T$ **do**
    for $i = 1, \ldots, n$ **do**
      if $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$ **then**
        $\theta = \theta + y^{(i)} x^{(i)}$
  return θ

We also explored graphically, what it did. It went something like this

- Firstly θ vector was initialised to be 0 (vector) : $\theta = 0$
- The we decided a T ie number of times we will iterate through the whole training set

for t = 1, 2 .... T do

- Then we one by one went to each of the example in train set

for i = 1, 2, 3 ...., n do

- we asked whether the θ that we have till now correctly classifies the ith point

If $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$ then

If YES
This means the classification is not correct and we update
$\theta = \theta + y^{(i)} x^{(i)}$

If NO
This means the classification is correct
we do nothing

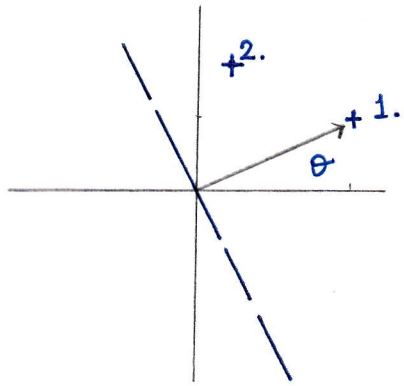
This algorithm had an intuitive graphical picture. And that was because $y^{(i)} x^{(i)}$: was the vector from origin to ith point. It pointed in the same direction as ith point if the label was +1. It pointed in the opposite (but parallel) direction as ith point if the label was −1.

The graphical/geometric picture was follows

- We started with the zero vector $\theta = 0$
- We got to the 1st point. $(y^{(1)}(\theta \cdot x^{(1)}))$ is always 0 for the 1st point so

  $\rightarrow \theta = 0 + y^{(1)} x^{(1)}$

  Ie we draw a vector from origin to the point (here $y^{(1)} = +1$)



- We then go to the 2nd point and check if $y^{(2)}(\theta \cdot x^{(2)}) \leq 0$ according to above θ

  Here it's correctly classified (see figure)

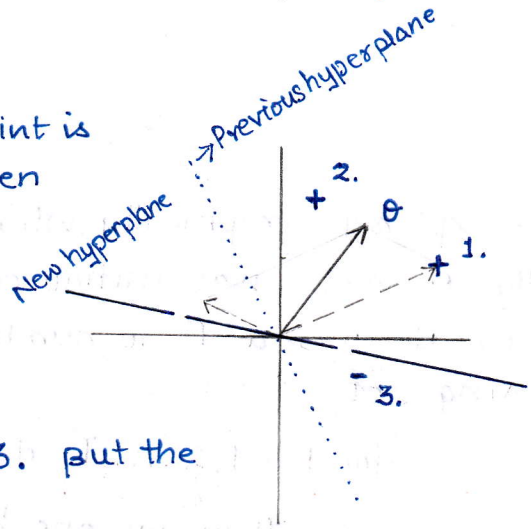
  So we move on to the 3rd point.

- The 3rd point is a -1 Label point. The point is misclassified by the current θ. as seen

  So

  $\rightarrow \theta = \theta_{old} + y^{(3)} x^{(3)}$

  $= y^{(1)} x^{(1)} + y^{(3)} x^{(3)}$



$y^{(3)} x^{(3)}$ is the vector from origin to 3. but the label $y^{(3)} = -1$ so it's away from 3.

And this is basically how the algorithm proceeds....

It goes through each example T times (because of the outer loop)

and asks the ith point if $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$

If yes then it adds $y^{(i)} x^{(i)}$ to the previous vector θ

If no then it it does nothing because point correctly classified

The it goes to the next point

I am emphasizing these points because of the next thing I am about to discuss.

We will do the same computations, the same steps and same procedure but with a different point of view

Lets look at it this way

The θ vector that is being made by the algorithm is either adding $y^{(i)} x^{(i)}$ to the previous θ vector if ith point is misclassified by the current θ, or doing nothing

Suppose for the sake of example, I decided $T = 3$ ie I will iterate 3 times over the <u>whole</u> dataset (training)
And suppose the training set has only 4 examples ie $n = 4$

<u>Suppose</u> the algorithm proceeds such that

→ First θ will be initialised 0

→ θ will update to $\theta = y^{(1)} x^{(1)}$

Now suppose as the algorithm progresses, only the 4th example is misclassified. (It sequentially proceeded from 1→2→3→4)

→ θ will update to $\theta = y^{(1)} x^{(1)} + y^{(4)} x^{(4)}$

Now the T's loop will kick in and the algorithm again loops from point 1 to point 4 <u>sequentially</u>

Suppose this time 2nd and 4th point contributed. The 4th point was classified properly earlier, but it may so happen that after the 2nd point update on <u>THIS</u> iteration (of T loop), 4th point is again misclassified. So by the end of this loop I will have

$$\theta = \theta_{old} + y^{(2)} x^{(2)} + y^{(4)} x^{(4)}$$

subst. θ_{old} from above

$$\theta = y^{(1)} x^{(1)} + y^{(4)} x^{(4)} + y^{(2)} x^{(2)} + y^{(4)} x^{(4)}$$

$$\theta = y^{(1)} x^{(1)} + y^{(2)} x^{(2)} + 2 * y^{(4)} x^{(4)}$$

Now T's loop again kick in for the 3rd iteration over train set

Suppose this time sequentially, the algorithm finds 1, 2, 3 points to be miss classified. By the end we will have

$$\theta = \theta_{old} + y^{(1)} x^{(1)} + y^{(2)} x^{(2)} + y^{(3)} x^{(3)}$$

Subst. θ_{old} from above

$$\theta = y^{(1)} x^{(1)} + y^{(2)} x^{(2)} + 2 * y^{(4)} x^{(4)} + y^{(1)} x^{(1)} + y^{(2)} x^{(2)} + y^{(3)} x^{(3)}$$

$$\theta = 2 * y^{(1)} x^{(1)} + 2 * y^{(2)} x^{(2)} + y^{(3)} x^{(3)} + 2 * y^{(4)} x^{(4)}$$

Now the T loop stops and we have our θ

The point is for large enough T and linearly seperable set, we will definitely find a proper θ classifying all points correctly

But that's not the point I wanna make here

The thing I want you to note is this θ vector which is being updated frequently is a made up of elements like $y^{(j)}x^{(j)}$ for $j = \{1, 2, \ldots n\}$

Ie at any point in time θ can be written as

$$\theta = \alpha_1 y^{(1)} x^{(1)} + \alpha_2 y^{(2)} x^{(2)} + \ldots\ldots + \alpha_n y^{(n)} x^{(n)}$$

$$\text{Ie } \theta = \sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)}$$

These α's are non negative integers. We will talk about them, but first note that if

$\alpha_1 \; \alpha_2 \; \alpha_3 \ldots\ldots \; \alpha_n = 0$ Then the θ vector above will just be 0 vector

So saying $\theta = 0$ for initialization is like saying that $\alpha_1 = \alpha_2 = \alpha_3 = \cdots = \alpha_n = 0$ for initializing

Now what does this α_j mean?
That's simple. It's the number of times the vector $y^{(j)}x^{(j)}$ contributes to the total vector θ upto that point of time in the algorithm
So when we check if jth point is misclassified ie: $(y^{(j)}(\theta \cdot x^{(j)})) \leq 0$ and it comes out to be TRUE.
Instead of updating θ by adding $y^{(j)}x^{(j)}$ we can <u>equivalently</u> say that $\alpha_j = \alpha_j + 1$ now

More clearly suppose Before the check θ Looked like

$$\theta_{old} = \alpha_1 y^{(1)} x^{(1)} + \alpha_2 y^{(2)} x^{(2)} + \ldots + \alpha_j y^{(j)} x^{(j)} + \ldots\ldots + \alpha_n y^{(n)} x^{(n)}$$

we checked if $y^{(j)}(\theta \cdot x^{(j)}) \leq 0$ & its TRUE THEN we update according to perceptron algorithim

$\theta_{new} = \theta_{old} + y^{(j)} x^{(j)}$. Subst θ_{old} from above

$$= \alpha_1 y^{(1)} x^{(1)} + \alpha_2 y^{(2)} x^{(2)} + \ldots + (\alpha_j + 1) y^{(j)} x^{(j)} + \ldots + \alpha_n y^{(n)} x^{(n)}$$

So this is equivalent to saying that update $\alpha_j = \alpha_j + 1$ & then use θ as $\sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)}$

What I am trying to do is write this perceptron algorithm, not in terms of θ but these α_j's and its coming up pretty well until now

So here goes the algorithm

- Write θ as $\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)}$

- To initialize $\theta = 0$

  Initialize $\alpha_1 = \alpha_2 = \alpha_3 = \cdots = \alpha_n = 0$

- The update rule

$$\text{if } \left(y^{(i)} (\theta \cdot x^{(i)}) \right) \leq 0 \quad \xleftrightarrow{\text{eqvn.}} \quad \text{if } \left(y^{(i)} \left(\sum_{j=1}^{n} \alpha_j y^{(j)} x^{(j)} \right) \cdot x^{(i)} \right) \leq 0$$

↳ YES
↳ $\theta = \theta + y^{(i)} x^{(i)}$ $\xleftrightarrow{\text{eqvn.}}$ ↳ YES
↳ $\alpha_i = \alpha_i + 1$

↳ NO
↳ Do nothing $\xleftrightarrow{\text{eqvn.}}$ ↳ NO
↳ Do nothing

For the example we saw earlier where $T = 3$ and $n = 4$

$\alpha_1 = \alpha_2 = \alpha_3 = \alpha_4 = 0$ # Initialization

1'st T's iteration

points 1 and 4 were misclassified

$\alpha_1 = 1$, $\alpha_2 = 0$, $\alpha_3 = 0$, $\alpha_4 = 1$

2nd T's iteration

points 2 and 4 were misclassified

$\alpha_1 = 1$, $\alpha_2 = 1$, $\alpha_3 = 0$, $\alpha_4 = 1+1 = 2$

3rd T's iteration

point 1, 2, and 3 were misclassified

$\alpha_1 = 1+1 = 2$, $\alpha_2 = 1+1 = 2$, $\alpha_3 = 1$, $\alpha_4 = 2$

So finally we have

$\alpha_1 = 2$, $\alpha_2 = 2$, $\alpha_3 = 1$, $\alpha_4 = 2$

The final vector is

$$\theta = 2 * x^{(1)} y^{(1)} + 2 * x^{(2)} y^{(2)} + x^{(3)} y^{(4)} + 2 * x^{(4)} y^{(4)}$$

Same as before. because the algorithm is still <u>exactly</u> the same, this is just a different way of doing it

But why are we doing. Why go all the way out to write this perceptron algorithm a little differently when we know the algorithm very well already. What's the meaning of this. Why are we doing all this

Why are we still here? Just to suffer? Every night, I can feel my leg... And my arm... even my fingers... The body I've lost... the comrades I've lost... won't stop hurting... It's like they're all still there. You feel it, too, don't you? I'm gonna make them give back our past!

Anyways, the reason we are doing this will become clear once we move forward and apply this procedure to our new feature space

And applying the perceptron to the new feature space is simple we have just transformed our vector $x^{(i)}$ to $\phi(x^{(i)})$. We hope that we can find a proper classifier (ie θ) in this space, so the algorithm remains just the same. In the algo, instead of $x^{(i)}$ we write $\phi(x^{(i)})$ Here's how it looks

$$\theta = 0$$

$$\text{run through } i = 1, \ldots, n$$

$$\text{if } y^{(i)} \theta \cdot \phi(x^{(i)}) \leq 0$$

$$\theta \leftarrow \theta + y^{(i)} \phi(x^{(i)})$$

This is how the inner step will change

Now for some reason that will be clear after a little while, I want to write this with the little modification that we studied earlier. Again this is not a different procedure, but a different way of looking at the same procedure. We can write

$$\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi(x^{(j)})$$

We will use this expression for θ and update α_i by adding 1 if our check says that ith point is misclassified

Now comes the important part

The checking step in the perceptron is

$$y^{(i)} \; \theta \cdot \phi(x^{(i)}) \leq 0$$

Let's see how this looks when we substitute our expression of θ

$$\theta = \sum_{j=1}^{n} \alpha_j \, y^{(j)} \, \phi(x^{(j)})$$

So we have the check step equivalently written as

$$y^{(i)} \left[\sum_{j=1}^{n} \alpha_j \, y^{(j)} \, \phi(x^{(j)}) \right] \cdot \phi(x^{(i)})$$

this $\phi(x^{(i)})$ can be taken inside the summation as it does not depend on index j. So we have

$$y^{(i)} \sum_{j=1}^{n} \left[\alpha_j \, y^{(j)} \, \phi(x^{(j)}) \cdot \phi(x^{(i)}) \right]$$

Aaahaah !!!

We have already defined this thing called kernel

$$K(x, x') = \phi(x) \cdot \phi(x')$$

So we can write $\phi(x^{(j)}) \cdot \phi(x^{(i)})$ as $K(x^{(j)}, x^{(i)})$

And our check becomes

$$y^{(i)} \sum_{j=1}^{n} \alpha_j \, y^{(j)} \, K(x^{(j)}, x^{(i)}) \leq 0$$

And just by doing a little mathematical trick & defining this thingy called kernel we have reduced our mathematical complexity significantly.

And let me illustrate why

Let's say we have the original input/feature vector that is d dimensional

$x^{(i)} \in \mathbb{R}^d$ looks like $[x_1^{(i)}, x_2^{(i)}, x_3^{(i)} \ldots x_d^{(i)}]^T$

$x^{(j)} \in \mathbb{R}^d$ looks like $[x_1^{(j)}, x_2^{(j)}, x_3^{(j)} \ldots x_d^{(j)}]^T$

Suppose we want to do a feature transform of polynomial with degree p

$\phi(x)$ will have terms like

$1, x_1, x_2, x_3 \ldots\ldots x_d$	1st order	$d+1$ terms
$x_1^2, x_2^2 \ldots x_d^2, x_1x_2, x_1x_3 \ldots x_1x_d$	2nd order	d^2 terms
$x_1^3, x_2^3 \ldots x_1^2x_2 \ldots x_1x_2^2 \ldots$	3rd order	d^3 terms
$\vdots$		
$x_1^p, x_2^p, \ldots x_d^p, x_1x_2^{p-1} \ldots\ldots$	pth order	d^p terms

Now we are talking about large scale problems like $d = 10000$ so d^p will be 10000 times that of d^{p-1}

Therefore we are concerned with the order d^p to get a ball park idea of the computational complexity

Let's get a ball-park idea of how much the checking step

$$y^{(i)}(\theta \cdot \phi(x^{(i)})) \leq 0$$

approx. d^p dimensional approx d^p dimensional

So on each check (approximately) d^p multiplications are done

But if the same step is done using

$$y^{(i)} \sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x^{(i)}) \leq 0$$

If you remember for pth order the Kernel $K(x, x')$ has form

$$K(x, x') = (1 + \underset{d}{x} \cdot \underset{d}{x'})^p$$

d, d dimensional

So to find $K(x^{(i)}, x^{(j)})$ we need to d multiplications

and due to $\sum_{j=1}^{n}$ we need to do this n times

So a total of nd computations approximately

So if we use the normal perceptron : we have d^p

But if we use the Kernel formulation: we have nd

Obviously the Kernel formulation is much more efficient

And very clever indeed to reduce this complex problem to a much simpler framework.

So now its a good idea to study these things called Kernels, and that's what we will do next. Mathematicians have come up with some rules that eas the composition of a kernel.

▷ Kernel Compostion Rules

1. $K(x, x') = 1$ is a kernel

   If we take the transform $\phi(x) = 1$ and $\phi(x') = 1$ Then $\phi(x) \cdot \phi(x') = 1$

2. Let $f: \mathbb{R}^d \to \mathbb{R}$ & $K(x, x')$ is a kernel Then so is $\tilde{K}(x,x') = f(x) K(x,x') f(x')$

   Suppose $K(x, x') = \phi(x) \cdot \phi(x')$ where ϕ is some transform

   we can make a new transform $\phi_{new}(x) = f(x)\, \phi(x)$, note that $f(x)$, although a function, is real valued so $f(x)$ will be common element multiplied to each row entry of $\phi(x)$ to form $\phi_{new}(x)$. Similiarly

   $\phi_{new}(x') = f(x')\, \phi(x') = \phi(x')\, f(x')$

   Now we can dot the two $\phi_{new}(x) \cdot \phi_{new}(x') = f(x)\, \phi(x) \cdot \phi(x')\, f(x')$

   we know $\phi(x) \cdot \phi(x') = K(x, x')$

   Therefore $f(x)\, K(x, x')\, f(x')$ is also a kernel

3. If $K_1(x, x')$ and $K_2(x, x')$ are kernels then $K(x, x') = K_1(x, x') + K_2(x, x')$ is also a kernel

   Suppose $K_1(x, x') = \phi_1(x) \cdot \phi_1(x')$

   $K_2(x, x') = \phi_2(x) \cdot \phi_2(x)$

   If I now make a new transform that stacks the upper part of my vector with all the transformation functions in ϕ_1 and lower part of my vector with all the transformation functions in ϕ_2. that looks like

$$\phi(x) = \begin{bmatrix} \phi_1(x) \\ \phi_2(x) \end{bmatrix}$$ Then

when I take the dot product $\phi(x) \cdot \phi(x')$ it looks like

$$\phi_1(x) \cdot \phi_1(x') + \phi_2(x) \cdot \phi_2(x')$$

$$K_1(x, x') + K_2(x, x')$$

is also a kernel.

4. If $K_1(x,x')$ and $K_2(x,x')$ are kernels then $K(x,x') = K_1(x,x')K_2(x,x')$ is also a kernel

The proof can be generalised from what I show below

$K_1(x,x')$ Let $K_1(x,x')$ be made of

$$= \begin{bmatrix} \phi_1(x) & \phi_2(x) \end{bmatrix} \begin{bmatrix} \phi_1(x') \\ \phi_2(x') \end{bmatrix} = \phi_1(x)\phi_1(x') + \phi_2(x)\phi_2(x')$$

Lets $K_2(x,x')$ be made of

$$= \begin{bmatrix} \phi_3(x) & \phi_4(x) \end{bmatrix} \begin{bmatrix} \phi_3(x') \\ \phi_4(x') \end{bmatrix} = \phi_3(x)\phi_3(x') + \phi_4(x)\phi_4(x')$$

Lets calculate

$$K_1(x,x')K_2(x,x') = \phi_1(x)\phi_1(x')\phi_3(x)\phi_3(x') + \phi_2(x)\phi_2(x')\phi_3(x)\phi_3(x')$$
$$\phi_1(x)\phi_1(x')\phi_4(x)\phi_4(x') + \phi_2(x)\phi_2(x')\phi_4(x)\phi_4(x')$$

Now if we define

$$\Phi(x) = \begin{bmatrix} \phi_1(x)\phi_3(x) \\ \phi_2(x)\phi_3(x) \\ \phi_1(x)\phi_4(x) \\ \phi_2(x)\phi_4(x) \end{bmatrix}$$

The $\Phi(x) \cdot \Phi(x')$ will be the expression above

But this expression is formed by the dot product of two transforms $\Phi(x)$ and $\Phi(x')$ therefore it's a kernel

Therefore $K_1(x,x')K_2(x,x')$ is also a kernel.

In summary the rules are

- Composition rules:
  1. $K(x,x') = 1$ is a kernel function.
  2. Let $f : \mathbb{R}^d \to \mathbb{R}$ and $K(x,x')$ is a kernel. Then so is $\tilde{K}(x,x') = f(x)K(x,x')f(x')$
  3. If $K_1(x,x')$ and $K_2(x,x')$ are kernels, then $K(x,x') = K_1(x,x') + K_2(x,x')$ is a kernel
  4. If $K_1(x,x')$ and $K_2(x,x')$ are kernels, then $K(x,x') = K_1(x,x')K_2(x,x')$ is a kernel

Now lets see an interesting Kernel that can be made using these rules

▷ Radial Basis Kernel

In order to make this I will do the calcuation for any general $x \in \mathbb{R}^d$ And in order to get a little feel, I would side by side do a calculation for $x \in \mathbb{R}^2$ looking like $[x_1 \; x_2]^T$ and x' looking like $[x_1' \; x_2']^T$
The result we will have is true for any d

i) we know $K_0(x,x') = 1$ is a kernel. If I take

$$\phi(x) = 1 \quad ; \quad \phi(x') = 1 \quad \text{then we can define}$$

$K_0(x,x') = \phi(x) \cdot \phi(x') = 1$ No problem

ii) Now we want to define $K_1(x,x') = \dfrac{x \cdot x'}{1!}$

we can do that, for eg in the case of $x \in \mathbb{R}^2$ we will have

$$\frac{x_1 x_1' + x_2 x_2'}{1!} \quad \text{so if we take } \phi(x) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \;\&\; \phi(x') = \begin{bmatrix} x_1' \\ x_2' \end{bmatrix}$$

(ie the identity transform)

we can make $\dfrac{x \cdot x'}{1!}$. This is also true for any d

so $K_1(x,x') = \dfrac{x \cdot x'}{1!}$ No problem

iii) Now we define $K_3(x,x') = \dfrac{(x \cdot x')^2}{2!}$

We can see how that looks for $x, x' \in \mathbb{R}^2$

$$\frac{(x \cdot x')^2}{2!} = \frac{(x_1 x_1' + x_2 x_2')}{2!} = \frac{x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2 x_1 x_2 x_1' x_2'}{2!}$$

$$\phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \end{bmatrix} \quad ; \quad \phi(x') = \begin{bmatrix} x_1'^2 \\ x_2'^2 \\ \sqrt{2} x_1' x_2' \end{bmatrix}$$

We can also do this for general dimension d so

$K_2(x,x') = \dfrac{(x \cdot x')^2}{2!}$ No prolem

iv) Let's similiarly define $K_3(x,x') = \frac{(x\cdot x')^3}{3!}$

For the case of $d=2$ it looks like

$$\frac{(x\cdot x')^3}{3!} = \frac{(x_1x_1' + x_2x_2')^3}{3!} = \frac{x_1^3x_1'^3 + x_2^3x_2'^3 + 3x_1^2x_2x_1'^2x_2' + 3x_1x_2^2x_1'x_2'^2}{3!}$$

$$\phi(x) = \begin{bmatrix} x_1^3 \\ x_2^3 \\ \sqrt{3}\,x_1^2x_2 \\ \sqrt{3}\,x_1x_2^2 \end{bmatrix}$$ will do our job, similiarly for $\phi(x')$

Hence $K_3(x,x') = \frac{(x\cdot x')^3}{3!}$ can be defined. No problem

By now you have the idea of where I am going with this I can for any general index (positive) j define the kernel.

$$K_j(x,x') = \frac{(x\cdot x')^j}{j!}$$

Now I want to make a kernel by adding all these kernels & I can do that because composition rule 13) allows me to do so

$$K_0(x,x') + K_1(x,x') + K_2(x,x') + \cdots\cdots$$

$$\equiv 1 + \frac{(x\cdot x')}{1!} + \frac{(x\cdot x')^2}{2!} + \frac{(x\cdot x')^3}{3!} + \cdots\cdots$$

is a kernel.

& we want to add these terms upto infinite terms.

Can we do this? Who knows. But we will do it

Our new kernel is $\sum_{j=0}^{\infty} K_j(x,x') = \sum_{j=0}^{\infty} \frac{(x\cdot x')^j}{j!}$

If you remember from Calculus

$$e^t = 1 + t + \frac{t^2}{2!} + \frac{t^3}{3!} + \frac{t^4}{4!} + \cdots\cdots$$

So our new Kernel is

$$\sum_{j=0}^{\infty} \frac{(x \cdot x')^j}{j!} = e^{x \cdot x'}$$

So our new kernel is $e^{x \cdot x'}$

Now composition rule 12> says if we have a function $f : \mathbb{R}^d \to \mathbb{R}$ and some kernel $K(x, x')$ then

$f(x)\ K(x, x')\ f(x')$ is also a kernel

So suppose we take

$$f(x) = \exp\left(-\frac{1}{2}\|x\|^2\right) \equiv e^{-\frac{\|x\|^2}{2}}$$

for eg in our case of $d = 2$ we had

$$f(x) = e^{-\frac{(x_1^2 + x_2^2)}{2}}$$

in general d case

$$f(x) = e^{\frac{-(x_1^2 + x_2^2 + x_3^2 + \cdots\cdots + x_d^2)}{2}}$$

If we use this $f(x)$ together with comp rule 12> and make the kernel

$$= e^{-\frac{\|x\|^2}{2}}\ e^{x \cdot x'}\ e^{-\frac{\|x'\|^2}{2}}$$

$$= e^{-\frac{1}{2}(\|x\|^2 - 2\,x \cdot x' + \|x'\|^2)}$$

$$= e^{-\frac{1}{2}(x^T x' - x^T x' - x'^T x + x'^T x')}$$

$$= e^{-\frac{1}{2}(x - x')^T (x - x')}$$

$$= e^{-\frac{1}{2}\|x - x'\|^2}$$

Note
$x^T x' = x'^T x$
because
$x, x' \in \mathbb{R}^d$

& this is our final Kernel

$$\boxed{e^{-\frac{1}{2}\|x - x'\|^2}}$$

This is a very famous kernel called the radial basis kernel

$$K(x,x') = \exp\left(-\frac{1}{2}\|x-x\|^2\right)$$

& it is famous for a very special reason

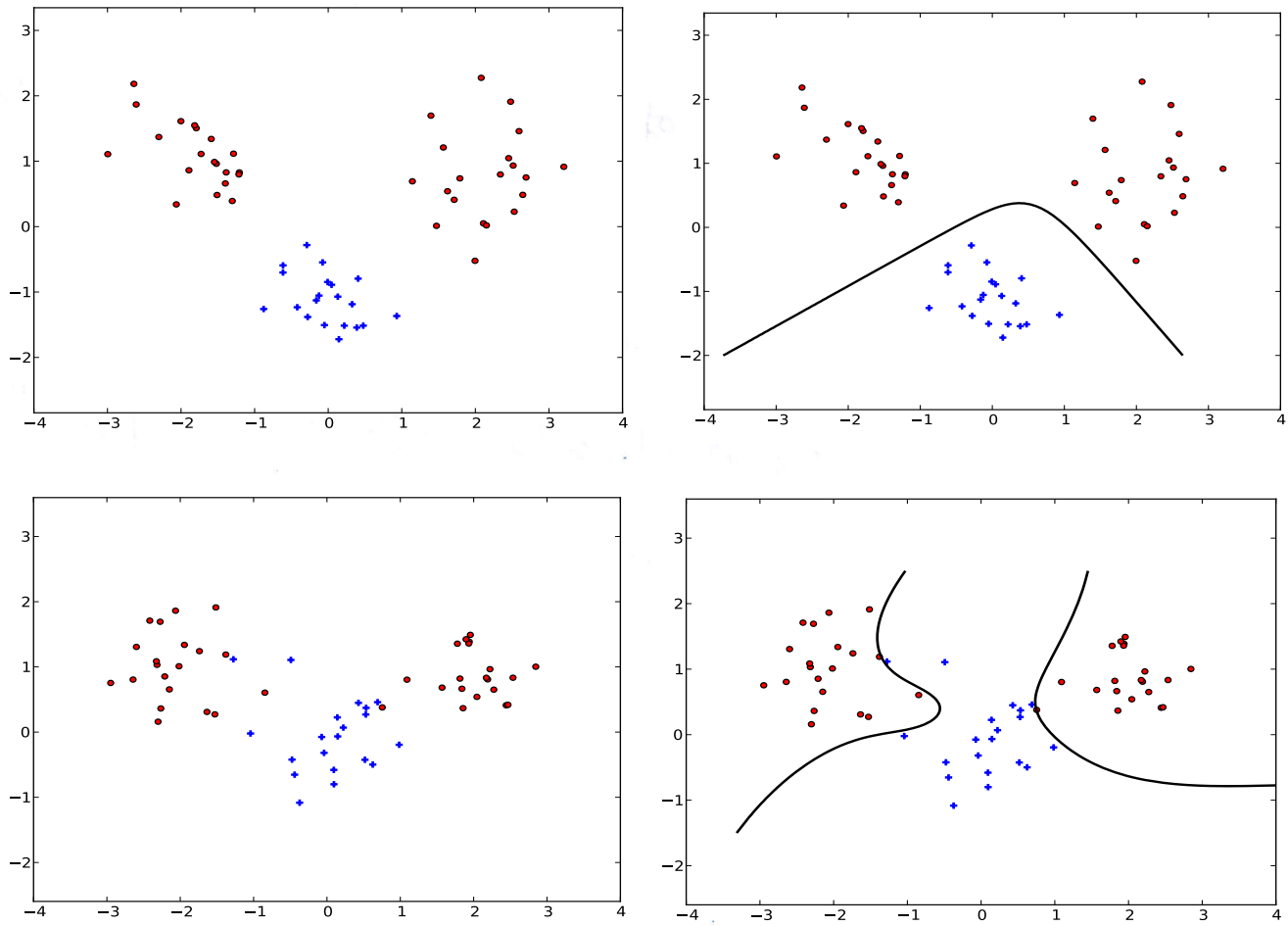
It is infinitely powerful. It will do whatever it takes to classify all points ~~of~~ of train data correctly.

Let me take a brief detour here to explain the wise pun above. There is this movie series called Avengers. and in a former part of the story, the antagonist called Thanos, wipes half of world's population of the Earth (maybe Universe) with the snap of his finger. He was able to do this because of the infinity stones he had. In the later part of the series, the remaining Avengers do 'whatever it takes' to set things right. The radial basis kernel has nothing to do with Avengers yet I used the wordplay of it being infinitely powerful and doing whatever it takes. This resulted in a chuckle from your side, at my wise joke. Let's get back to the topic.

We saw how the $K(x,x')$ was formed. It was made from an infinite series by adding polynomial powers of $x \cdot x'$ infinite times. What this means is. If we apply perceptron on the feature space that is transformed using the radial basis kernel, we will always find a correct solution, ie a set of boundaries that will classify ALL points correctly.

Because it has infinite power. No data is strong enough for it

Lets see Prof T.'s examples



The equation of the boundary is as we know

$$\theta \cdot \phi(x) = 0$$

and the classifier is $\text{sign}(\theta \cdot \phi(x))$

but we already had decided to use the new way of thinking when

$$\theta = \sum_{j=1}^{n} \alpha_j y^{(j)} \phi(x^{(j)})$$

So our decision boundary will be

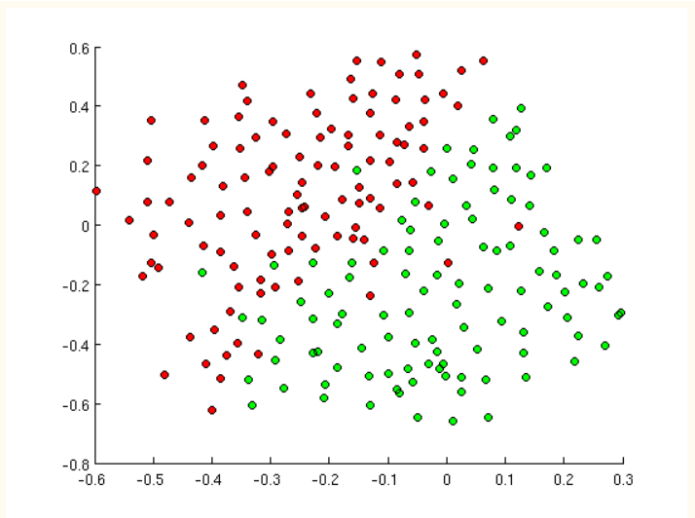
$$\{x : \sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x) \quad 0\}$$

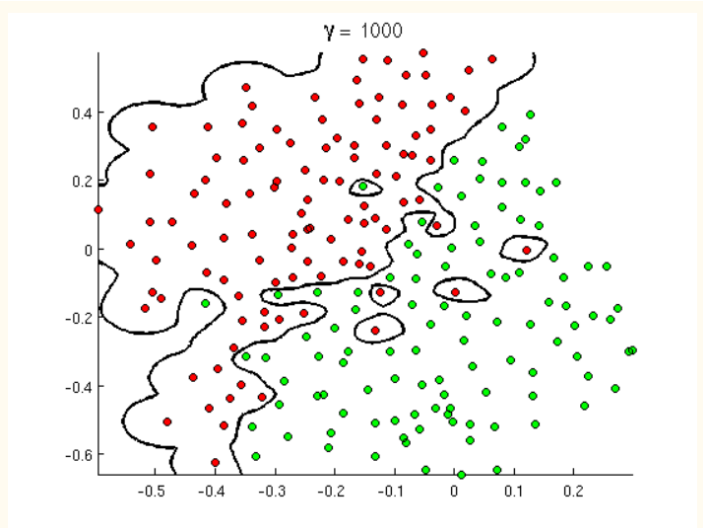
& decision rule will be

$$h(x) = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y^{(j)} K(x^{(j)}, x) \right)$$

& these α_j's will be the coefficient converged by perceptron because we are using a radial basis kernel so we will always converge.

In fact always converging can be a problem too. Look at the problem of classification below taken from. **Prof Andrew Ng's course**
Inough we can divide the region into two parts classifying each point of train set correctly, we clearly are over fitting here. Hence we will need to tune the model.





I will not cover decision trees, because it's taught very loosely in the videos. Maybe later

FIN.