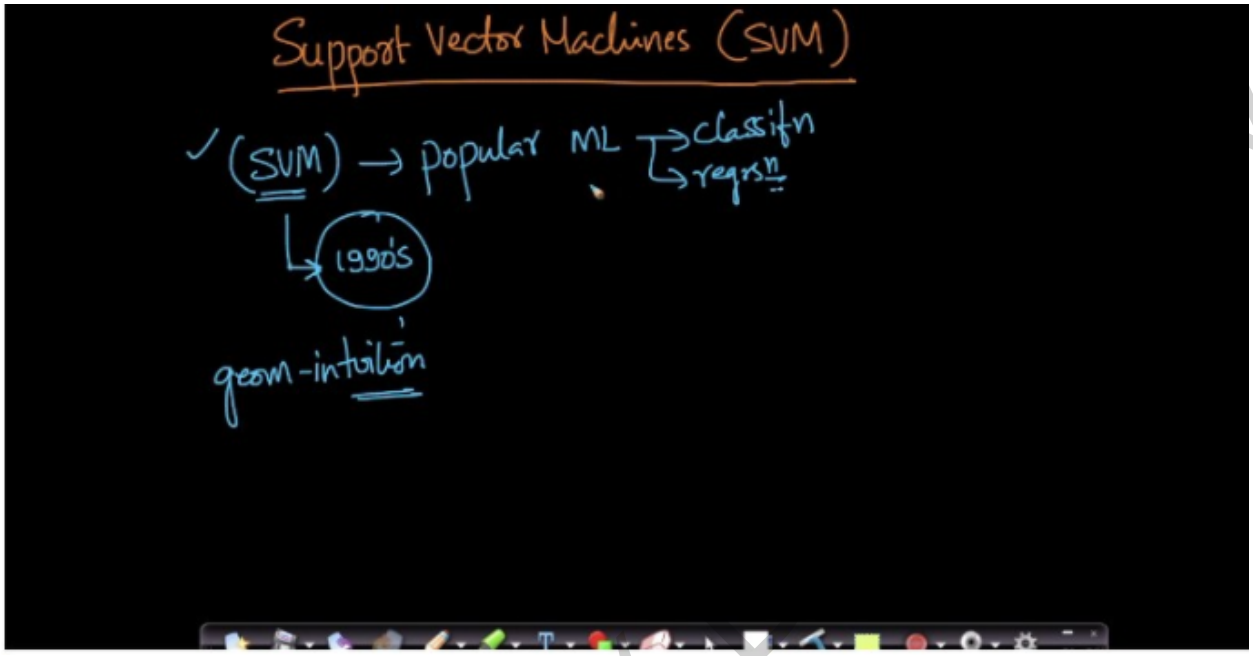
In this chapter we will look at an algorithm called Support Vector Machines(SVM).

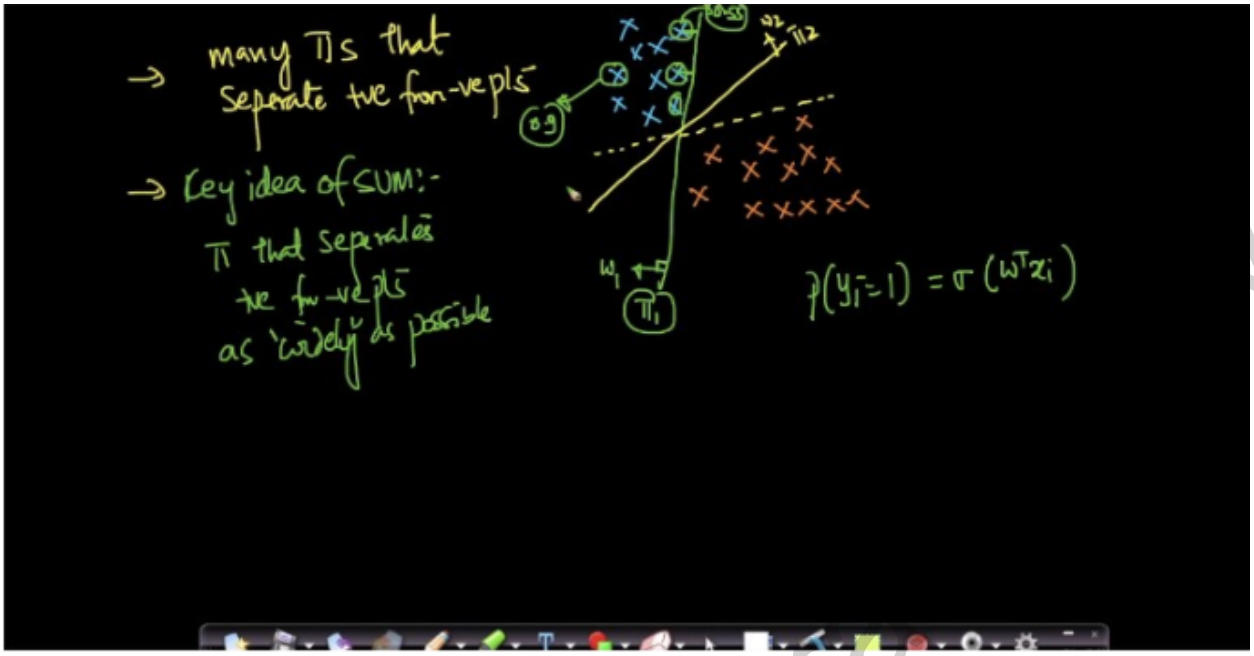
36.1 Geometric Intuition



Timestamp 1:01

Support Vector Machines(SVM) is a very popular algorithm. It was developed during the 1990's and it is still very popular. It can handle both regression and classification problems.

Let's first talk about the geometric intuition behind this algorithm.

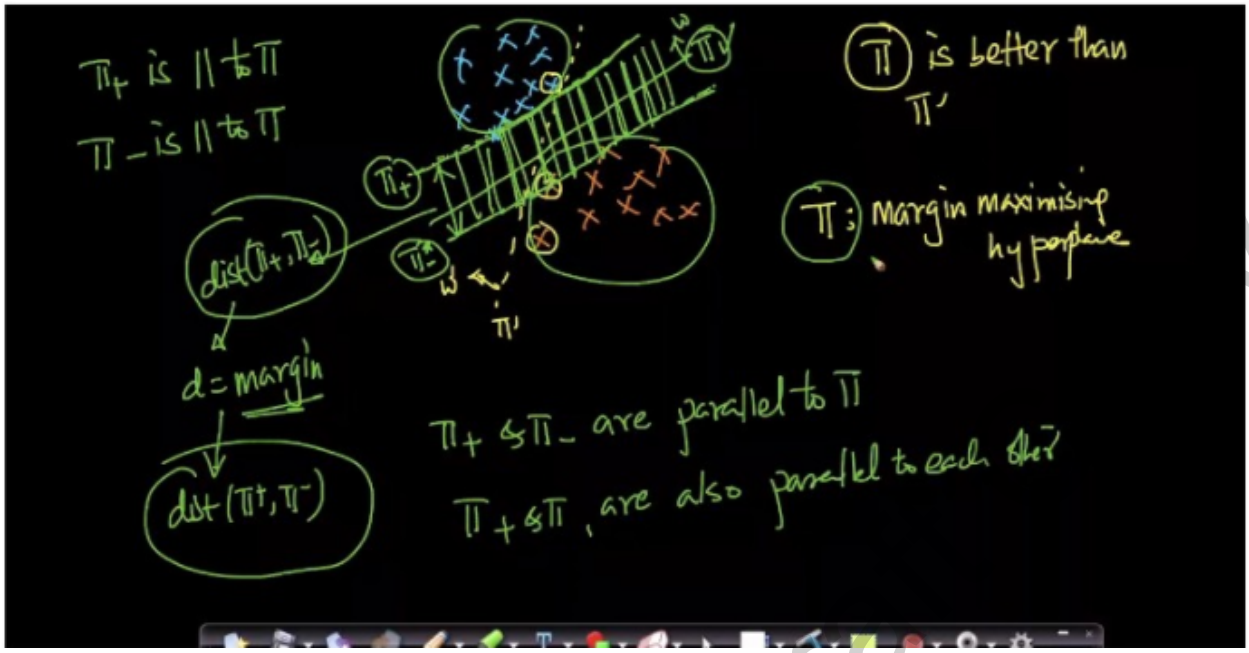


Timestamp 4:10

Suppose we have a dataset containing positive and negative points as shown in the image. Say the red crosses are negative points and the blue ones are positive. Now we want to find a hyperplane such that it separates these points. We will represent our hyperplane by π, each hyperplane has a normal vector, say w.

Now there can be multiple hyperplanes which can separate these points like π_1, π_2.etc as shown in the image above. Say we select π_1 as our separating hyperplane. If we do so we can see that some of the positive points are very close to the hyperplane $\pi_1$as shown in the image. Now for points closer to the hyperplane, algorithms like logistic regression will have very low confidence in its prediction whereas for points farther away from the hyperplane it will have a greater confidence in its prediction. So following this argument, we can conclude that in our dataset π_2 is a better hyperplane than π_1.

So the key idea of SVM is to find a hyperplane such that it separates the dataset as “widely” as possible.



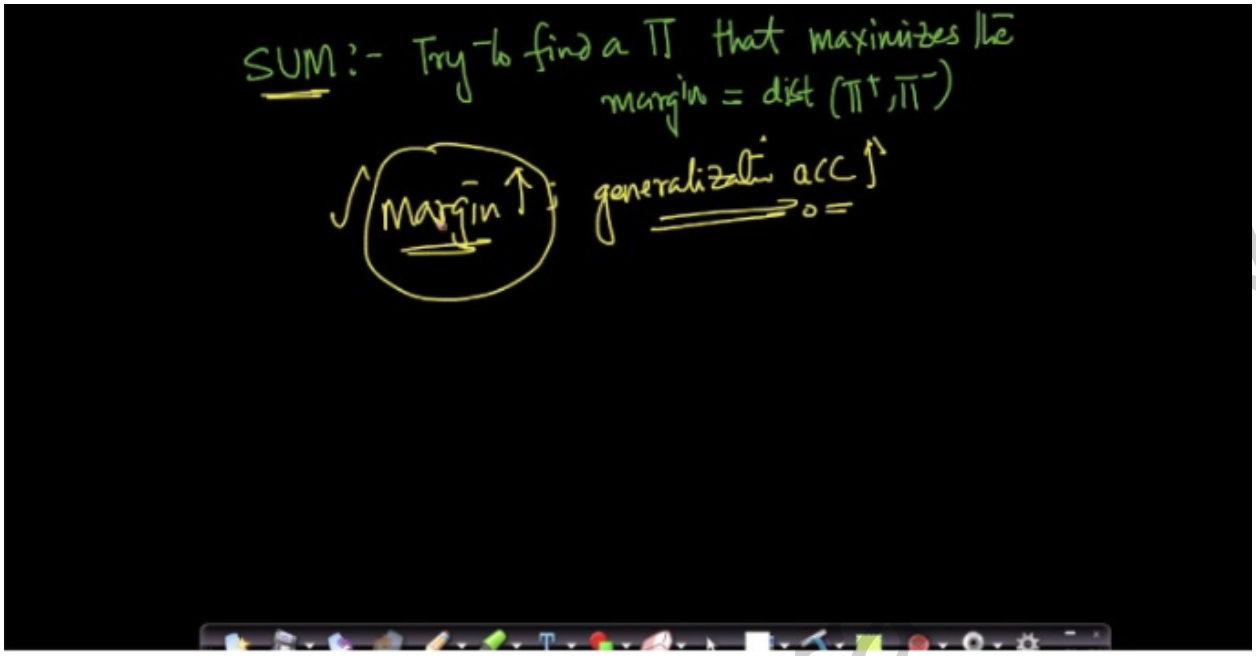
Timestamp 8:45

Now the question is how to define “widely”.

Here in the above image we have again taken the previous dataset and redrawn the hyperplanes. So from the intuition that we developed in the previous slide we know that π is better than π'. The hyperplane π is called the margin maximizing hyperplane.

Say we have drawn hyperplanes parallel to π towards the positive data points. The hyperplane which first intersects with a positive data point can be termed as positive hyperplane π_+. Similarly we can find a negative hyperplane π_- by going towards the negative data points. The distance between these two hyperplanes is called margin. Also note that π_+, π_-, π are all parallel to each other. We want to find a hyperplane such that it maximizes this margin.

If this distance is maximized then our positive and negative points are as far away from the separating hyperplane as possible.

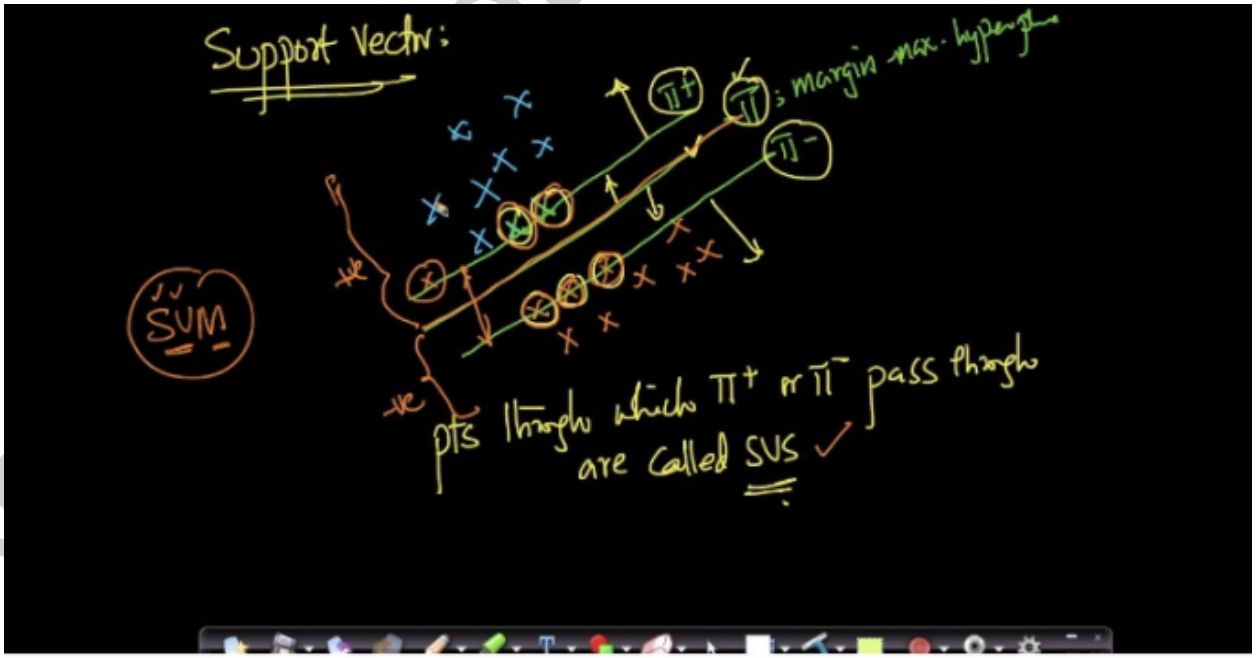


Timestamp 10:15

So in SVM's we essentially try to find a hyperplane π such that it maximizes the

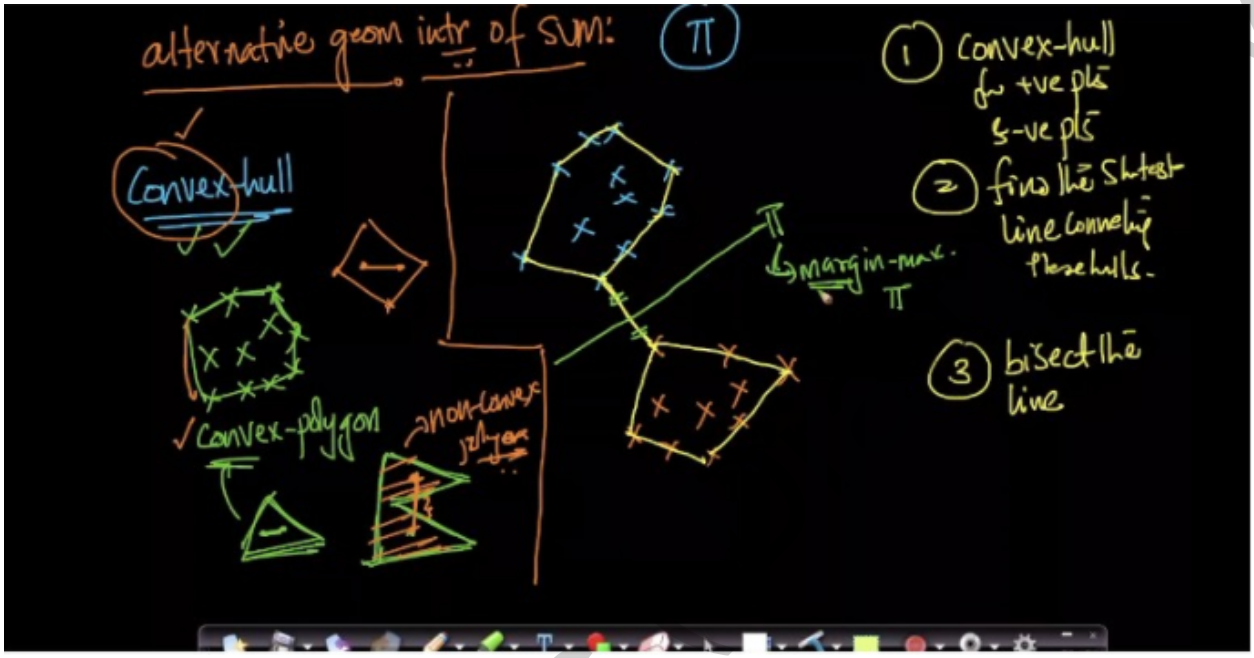
margin = distance(π_+ , π_-)

Also there is a nice theory which says that as the margin increases, accuracy on unseen data i.e. generalization accuracy increases.



Timestamp 13:08

The points through which our positive and negative hyperplanes pass are called support vectors. We can see the points which are passing through the hyperplanes in the above image, we have circled them. These points or support vectors are very important as we will see in later lectures.



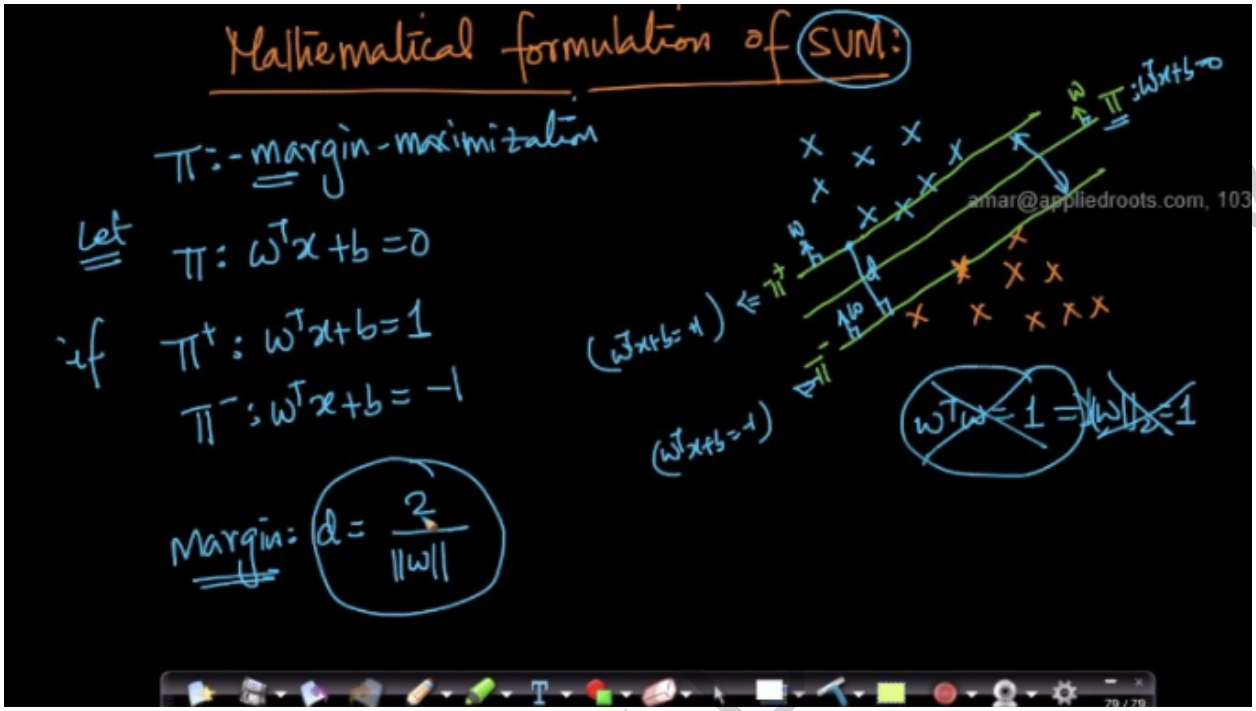
Timestamp 18:55

There is a nice alternative geometric interpretation of SVM. For this we need to define a few terms.
Convex Polygon - A convex polygon is a simple polygon in which no line segment connecting two points can ever go outside the polygon. For example, triangles as we have shown in the image above.
Convex Hull - A convex hull is the smallest convex polygon that contains all the data points. In order to draw a convex hull we need to draw a convex polygon such that it contains all the data points. It should also be the smallest so it may pass through some of the data points as shown in the above image.

Now coming back to SVM's. In order to find the margin maximizing hyperplane π :

1. Draw separate convex hulls for both positive and negative points as shown in the image above.
2. Find the smallest line connecting these convex hulls.
3. Bisect the smallest line that we found joining the convex hulls. This particular bisection is our margin maximizing hyperplane π .

36.2 Mathematical Derivation



Timestamp 3:17

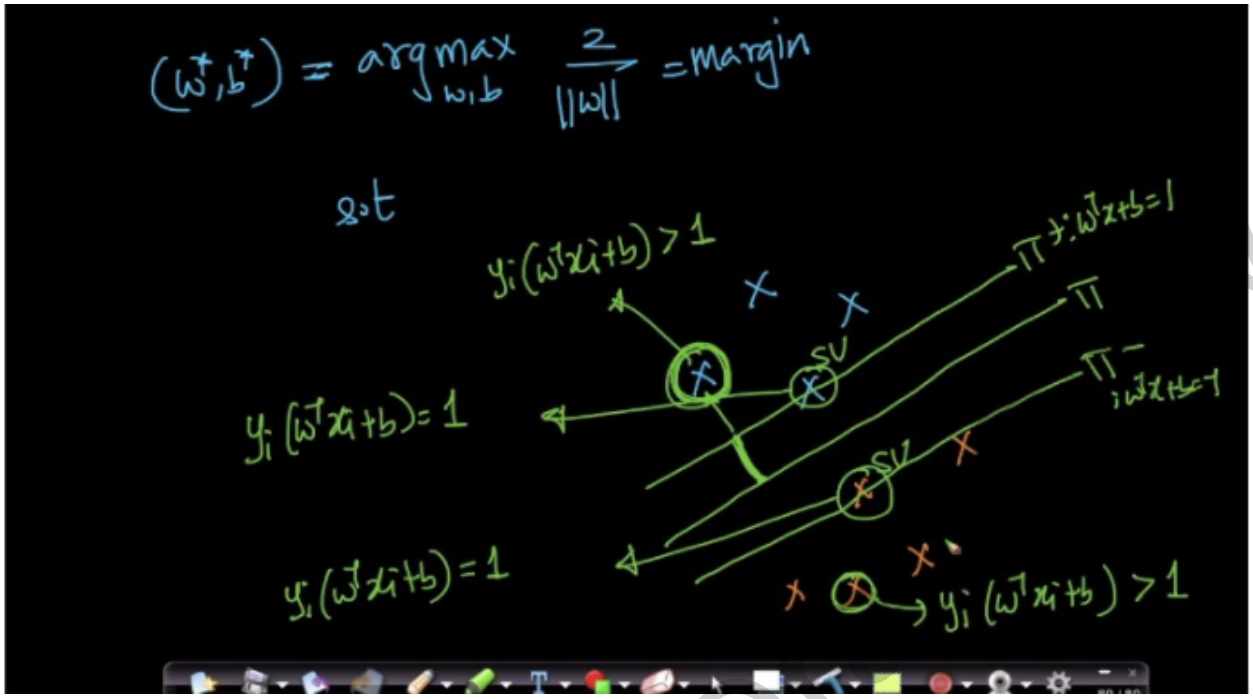
Here we will try to build the mathematical formulation of SVM.

Suppose we have a margin maximizing hyperplane π with a normal w. Also let, π_+ and π_- be positive and negative hyperplanes respectively. The normal w will also be perpendicular to both π_+ and π_- as they are both parallel to π .

Let π => $w^T x$ + b = 0

For now assume that π_+ => $w^T x$ + b = 1 and π_- => $w^T x$ + b = -1.

Now the distance between these hyperplanes will be
= 2 / || w ||
Also note that w is a normal vector not a unit vector.



Timestamp 7:44

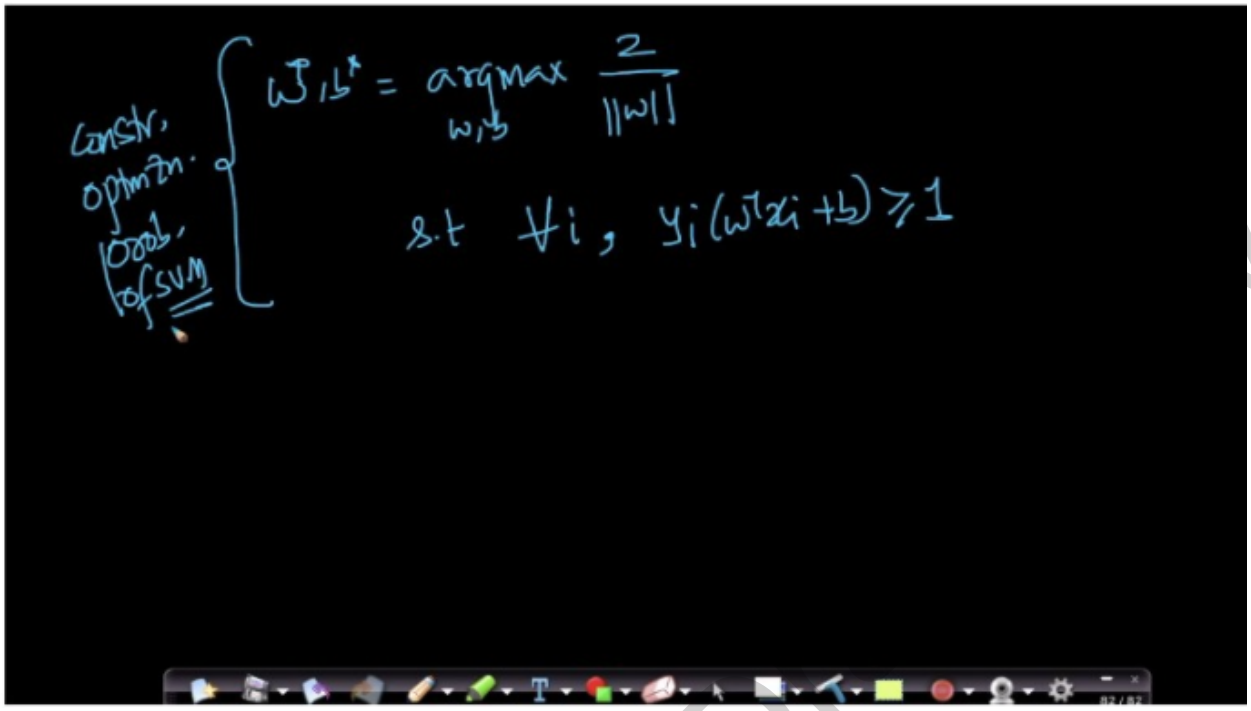
So we want to maximize this margin. We need to find a (w^*, b^*) such that,
(w^*, b^*) = $argmax_{(w,b)}$ 2 / || w ||

The above equation will have some constraints. Let's talk about those, say we represent our positive points by +1 and negative points by -1. So $y_i \epsilon$ { +1, -1}

Now for any positive support vector we have y_i = 1 and also the quantity $w^T x_i$ + b = 1. So the term y_i*($w^T x_i$ + b) becomes 1.

Also for any negative support vector we have y_i = -1 and also the quantity $w^T x_i$ + b = -1. So again the term y_i*($w^T x_i$ + b) becomes 1.

Also for positive points above the positive hyperplane i.e. in the direction of w we have y_i *($w^T x_i$ + b) > 1 and similarly for negative points below the negative hyperplane i.e. opposite to the direction of w we have y_i*($w^T x_i$ + b) > 1. These points can be referred to as non support vectors.



Timestamp 9:44

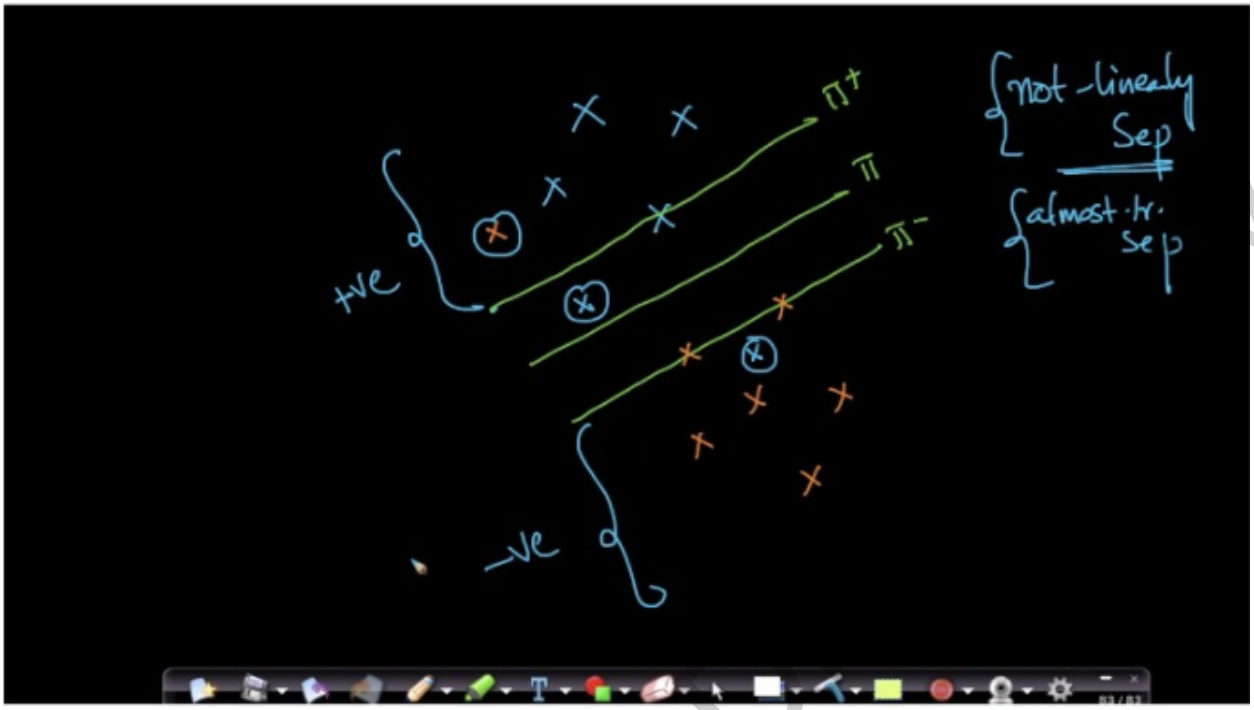
So, our final optimization problem is

$(w^*, b^*) = argmax_{(w,b)}$ 2 / || w ||,

s.t. ∀i , $y_i^*(w^T x_i$ + b) ≥ 1

Note that we have a total of n constraints for n data points, because we want the constraint to hold true for all data points. The equalities in the constraint are for the support vectors and the greater than part are for the non support vectors.

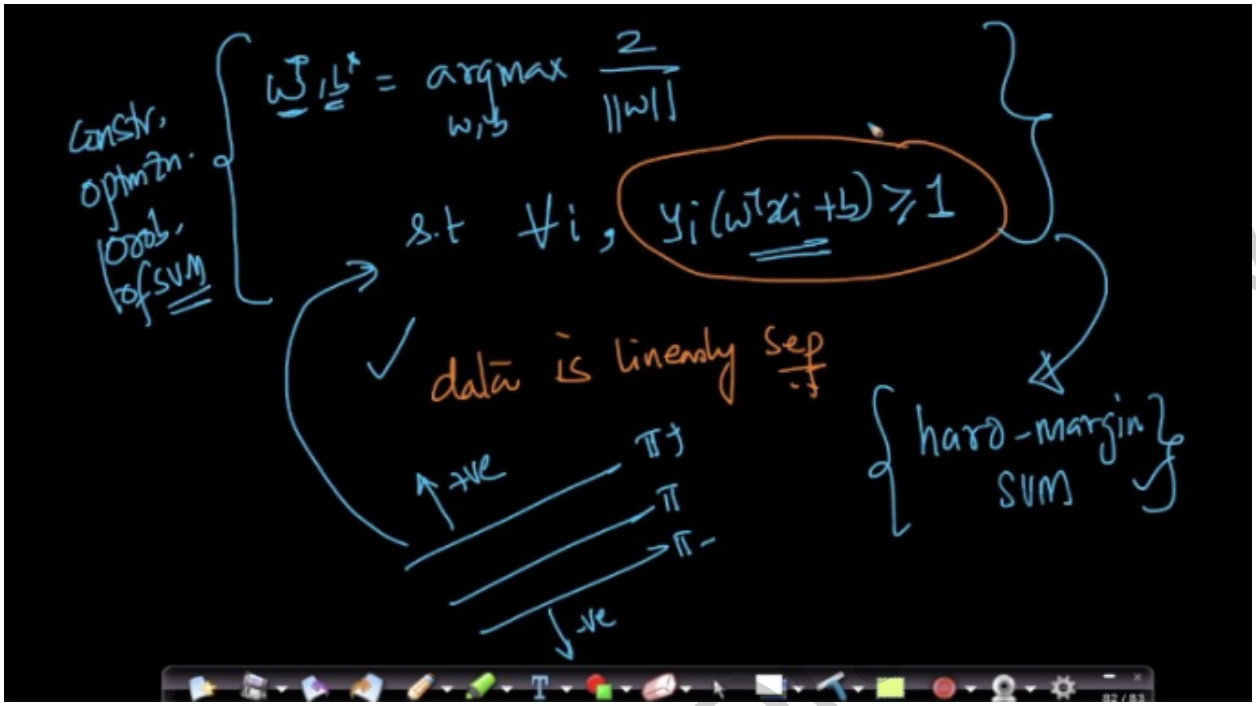
This type of optimization problems are called constraint optimization problems. We typically solve them using techniques like lagrangian multipliers.



Timestamp 12:32

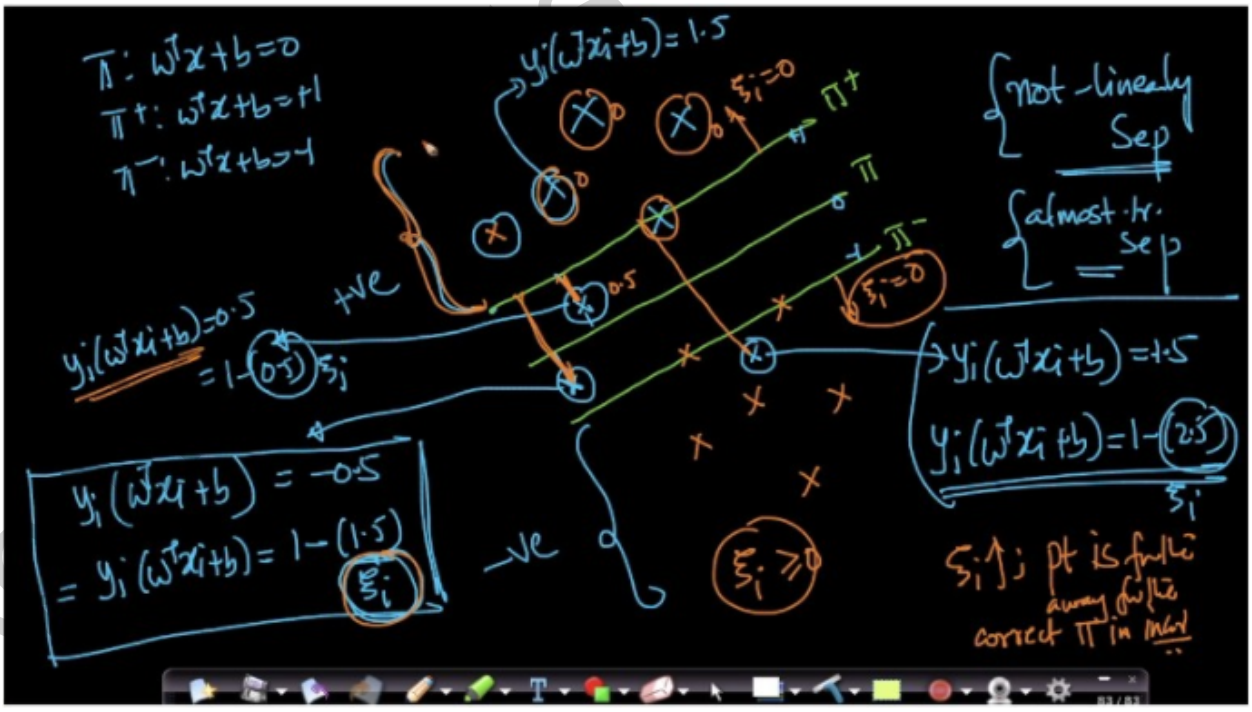
There is one issue with this optimization problem. Consider a dataset where the dataset is not linearly separable. As you can see from the image above there are some positive points which lie below the negative hyperplane and some negative points lie above the positive hyperplane. Such a dataset isn't linearly separable. It is almost linearly separable; only some points lie on the wrong side of the hyperplane.

So the constraints in our optimization problem i.e. $y_i^*(w^T x_i$ + b) ≥ 1 may not be satisfied for some of the data points.



Timestamp 13:48

This is the reason that this formulation is called hard margin SVM because it strictly requires the data points to be linearly separable i.e. all the data points should satisfy the constraint.



Timestamp 19:20

Now we need to handle the case where the points are not completely linearly separable. Consider a positive data point which lies on the negative side of the hyperplane as shown in the image above, say this point is -1.5 units away from the correct(positive) side of π . So in this case we can write

$y_i*(w^T x_i + b) = -1.5$

or, $y_i*(w^T x_i + b) = 1 - 2.5$

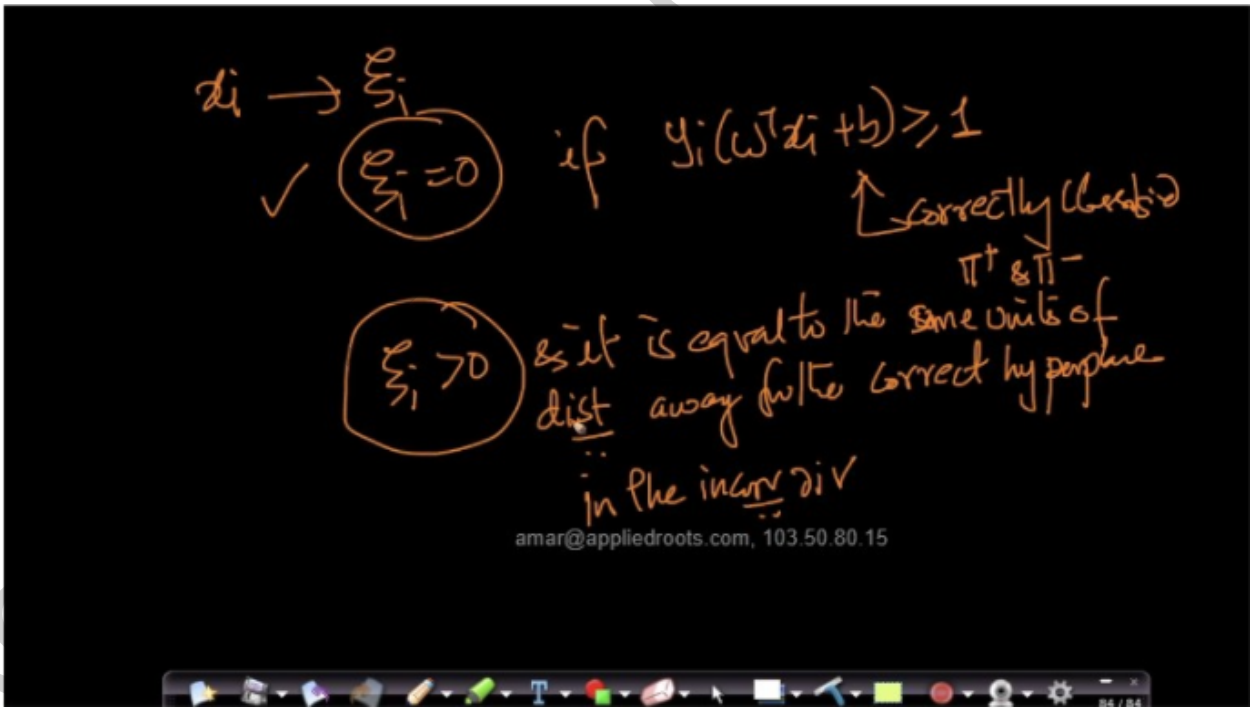
We are subtracting it from 1 because ideally we want $y_i*(w^T x_i + b) = 1$.

Similarly, say a positive data point lies between positive and negative hyperplane. Say it lies -0.5 units away from the correct(positive) side of π . So in this case we can write

$y_i*(w^T x_i + b) = -0.5$

or, $y_i*(w^T x_i + b) = 1 - 1.5$.

We can continue like this for all the misclassified points as shown in the image above and calculate the misclassified distance. In general, we write the misclassified distance as ξ(zeta).
For correctly classified points ξ is 0. For incorrectly classified points ξ > 0.
So in general $\xi \geq 0$.
A higher value of ξ indicates that the point is farther away from the correct hyperplane.



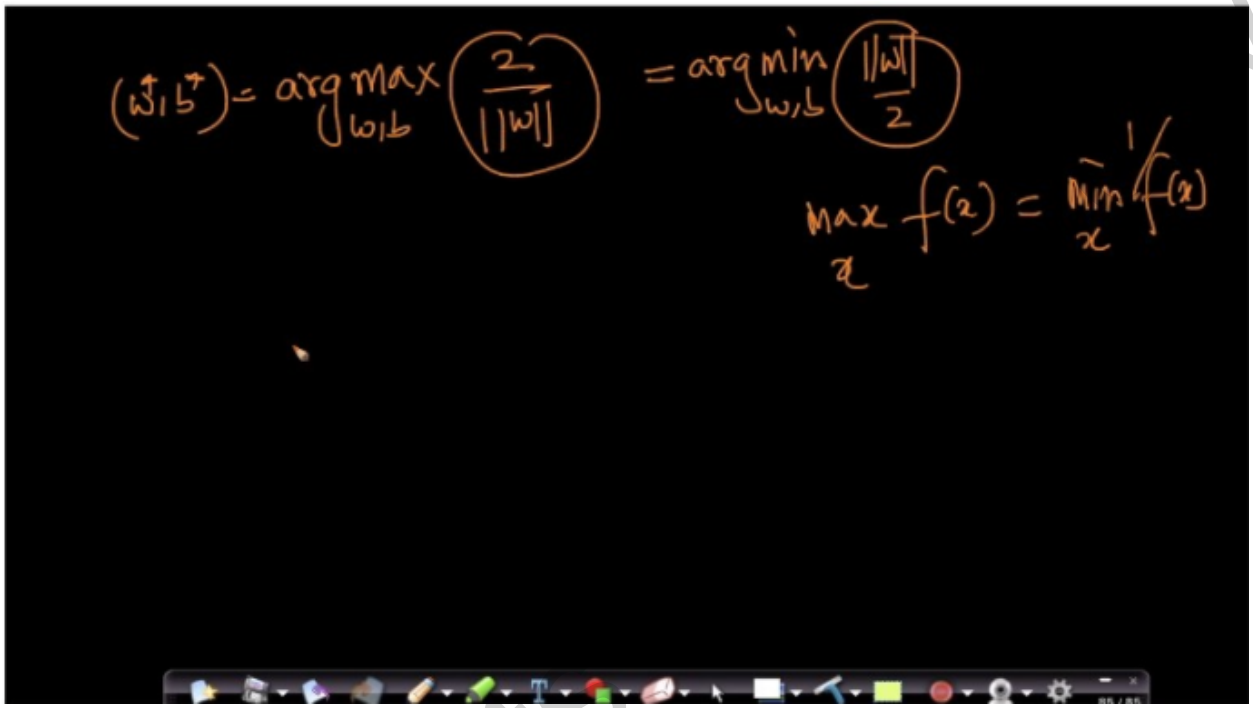
Timestamp 21:20

So to summarize,

For every datapoint x_i, we are getting a ξ_i, $x_i \rightarrow \xi_i$.

Now this $\xi_i = 0$, if $y_i^*(w^T x_i + b) \geq 1$ i.e. point is correctly classified w.r.t π_+ and π_-,

and $\xi_i > 0$ if the point is incorrectly classified and the magnitude of ξ_i represents the units by which it is away from the correct hyperplane.



Timestamp 22:11

Now we will reformulate our optimization problem.

Earlier we had,

$(w^*, b^*) = argmax_{(w,b)}$ 2 / || w ||,

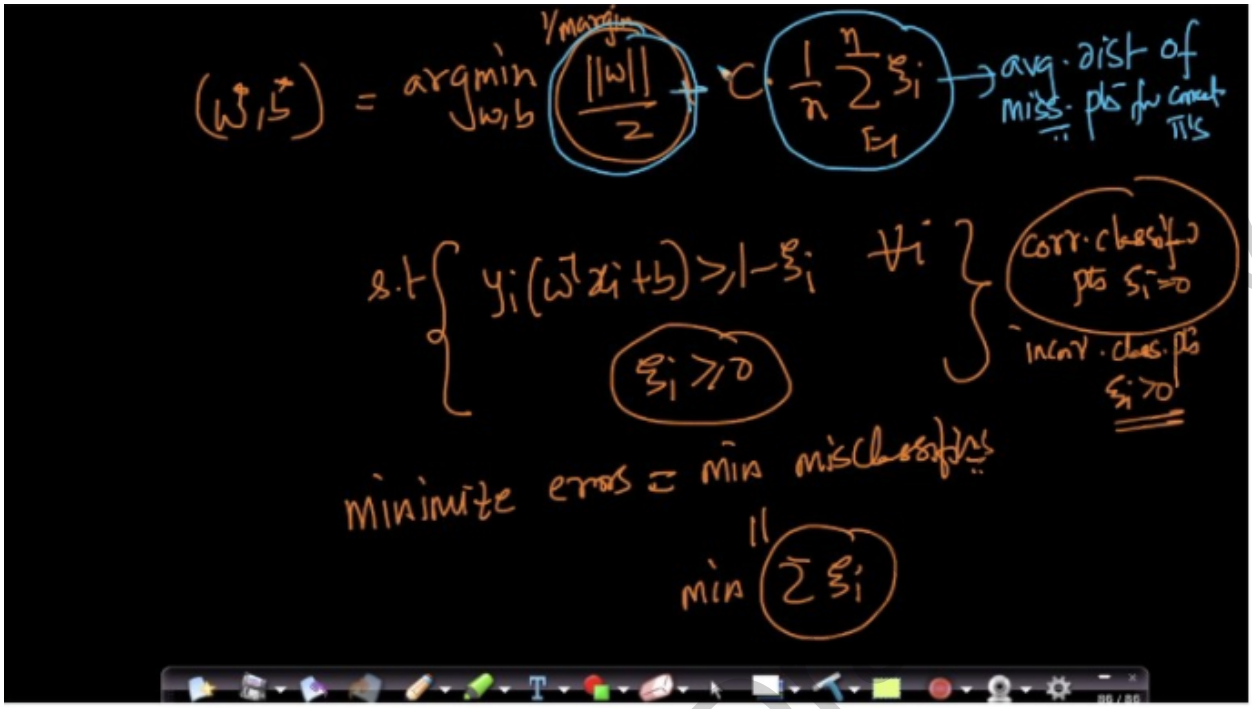
s.t. ∀i , $y_i^*(w^T x_i + b) \geq 1$

The above equation can also can be written as

$(w^*, b^*) = argmin_{(w,b)}$ || w || / 2,

s.t. ∀i , $y_i^*(w^T x_i + b) \geq 1$

This is because the max f(x) = min 1/f(x), when f(x) ≠0



Timestamp 25:14

So, our optimization problem is

$(w^*, b^*) = argmin_{(w,b)}$ || w || / 2 + C* 1/n $\sum_{i=1}^{n} \xi_i$

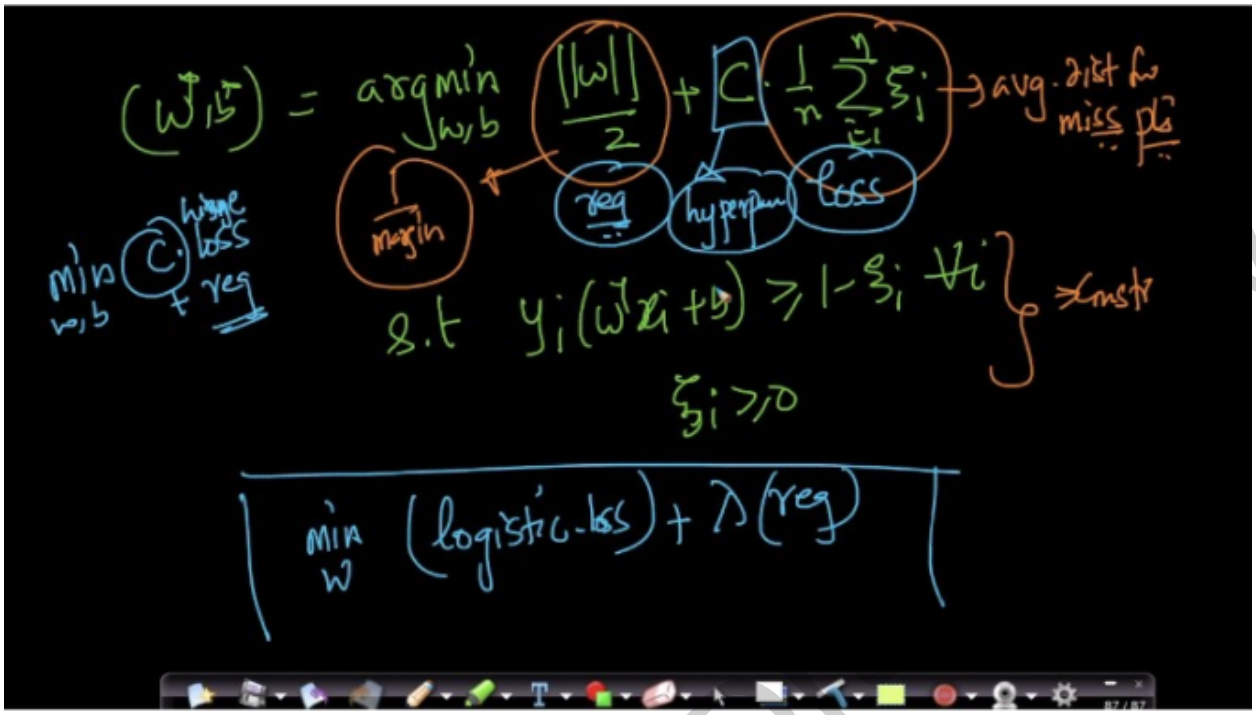
s.t. ∀i , y_i*($w^T x_i$ + b) ≥ 1 - ξ_i,

$\xi_i \geq 0$

where C is the hyperparameter and n is the no of data points.

1/n $\sum_{i=1}^{n} \xi_i$ => this term basically represents the average distance of the misclassified points, so we want to reduce it.

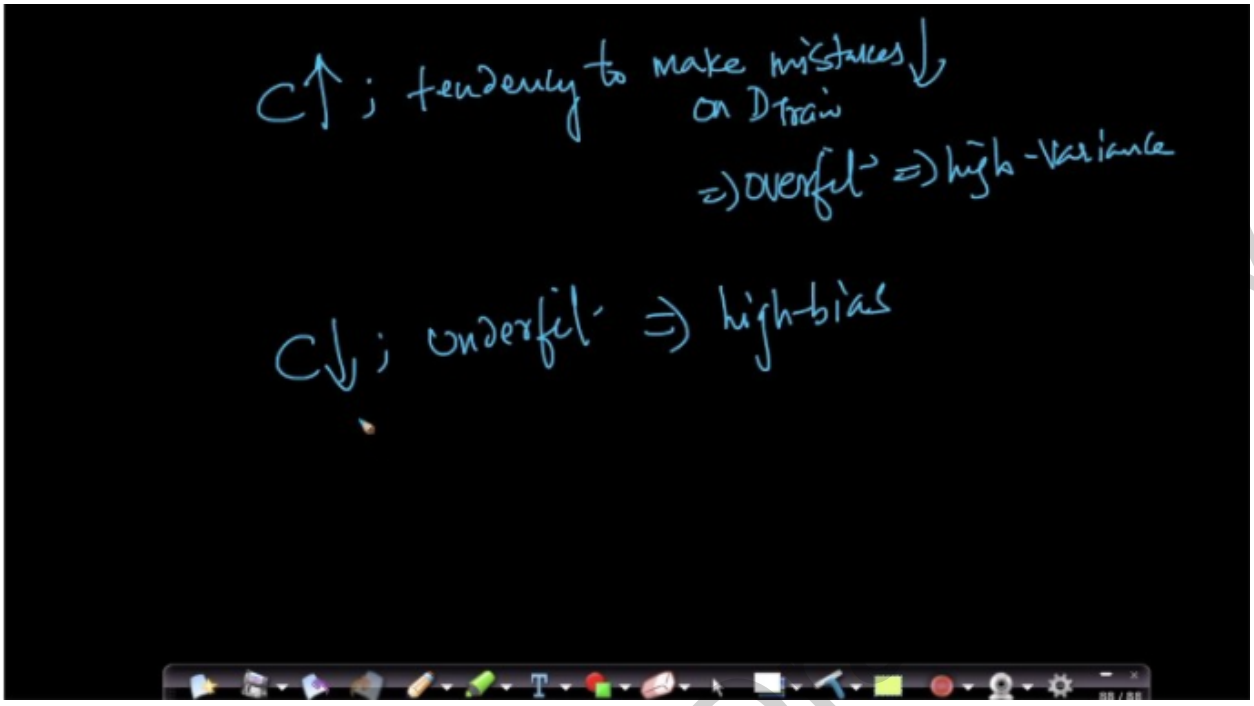
This formulation is called soft margin SVM. It allows errors unlike hard margin SVM, but asks to minimize them.



Timestamp 27:55

We can think of this term => 1/n $\sum_{i=1}^{n} \xi_i$ as our loss function because we want to reduce it. This loss is called hinge loss.
Also, || w || / 2 can be thought of as a regularizer. We have already seen a similar formulation while learning logistic regression as you can see in the image above.

So we can say that our formulation is something like this
$argmin_{(w,b)}$ C*hinge loss + regularizer, where C is the hyperparameter

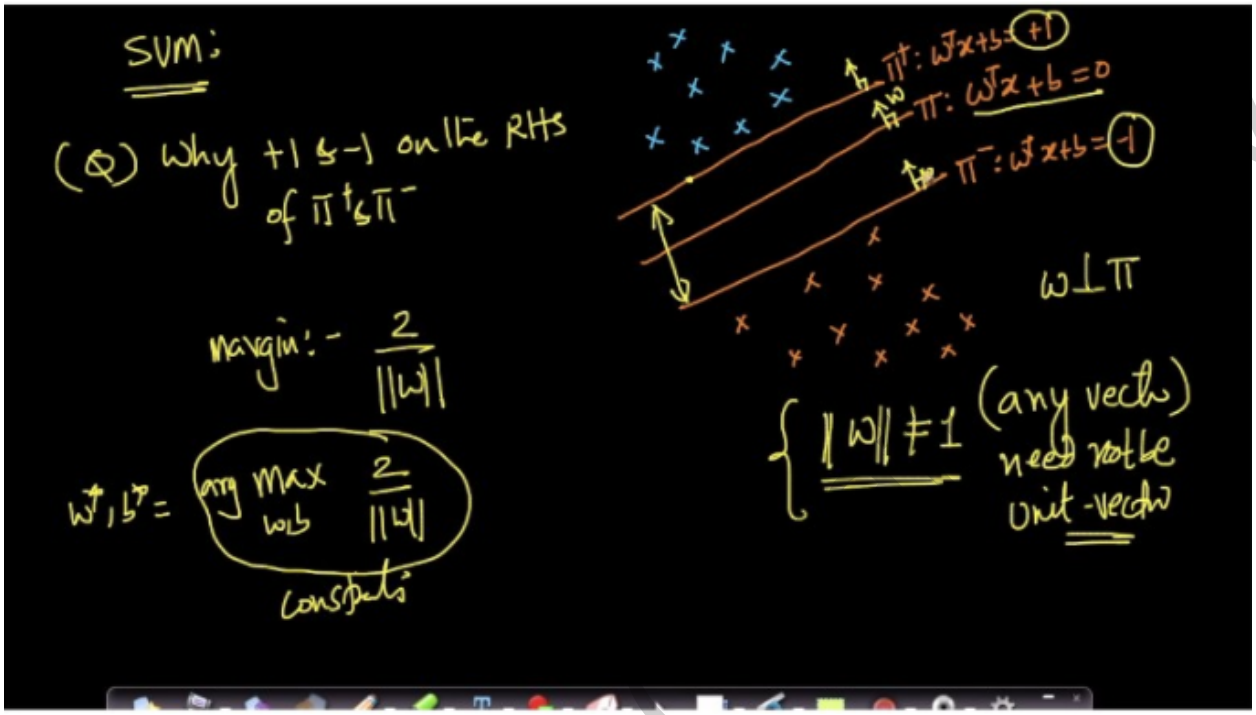


Timestamp 29:38

C is always ≥0.
If the value of C is high it means we cannot have a very high hinge loss. Meaning less misclassification. This results in overfitting => high variance.

If the value of C is low it means we can have a high hinge loss. Meaning more misclassification. This results in underfitting => high bias.

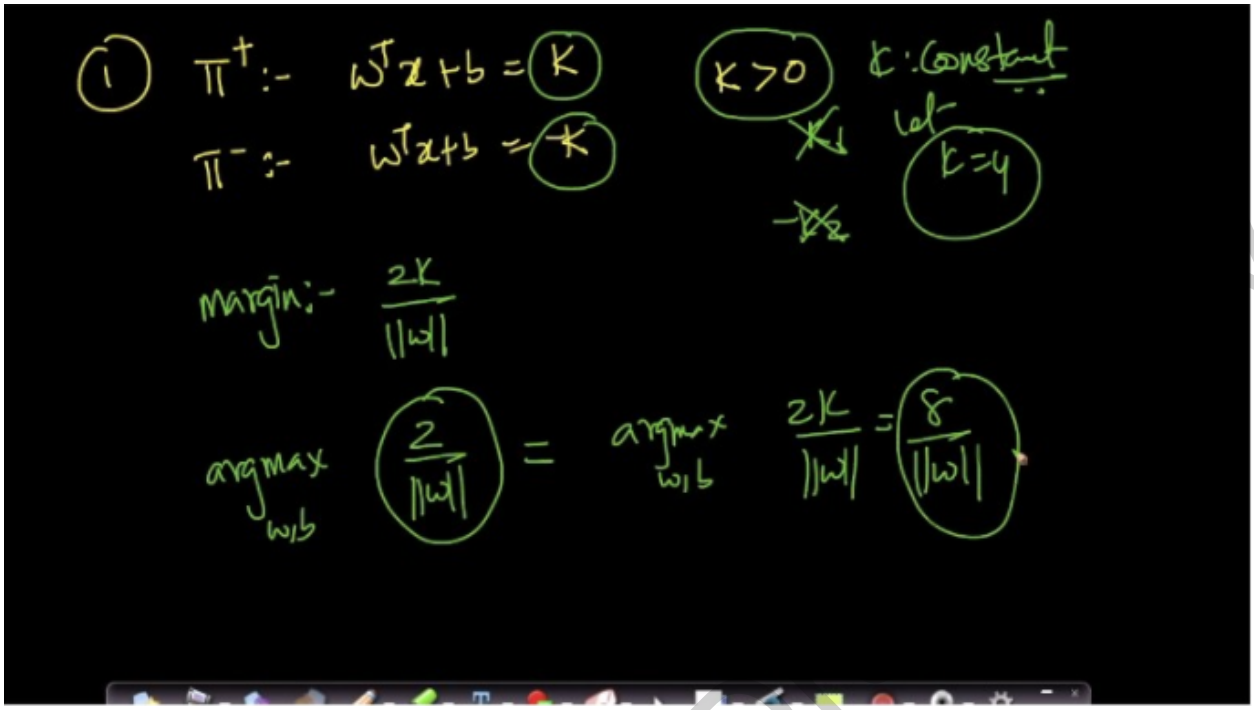
36.3 Why we take values +1 and and -1 for Support vector planes



Timestamp 2:43

Here we will discuss a question that was asked by one of our students.
Why are we taking -1 and +1 in the RHS of the negative and positive hyperplanes respectively?

Note that in previous lectures we have said w is not a unit vector. Also the margin between the hyperplanes was 2 / || w || and we wanted to maximize this based on some constraints.



Timestamp 4:49

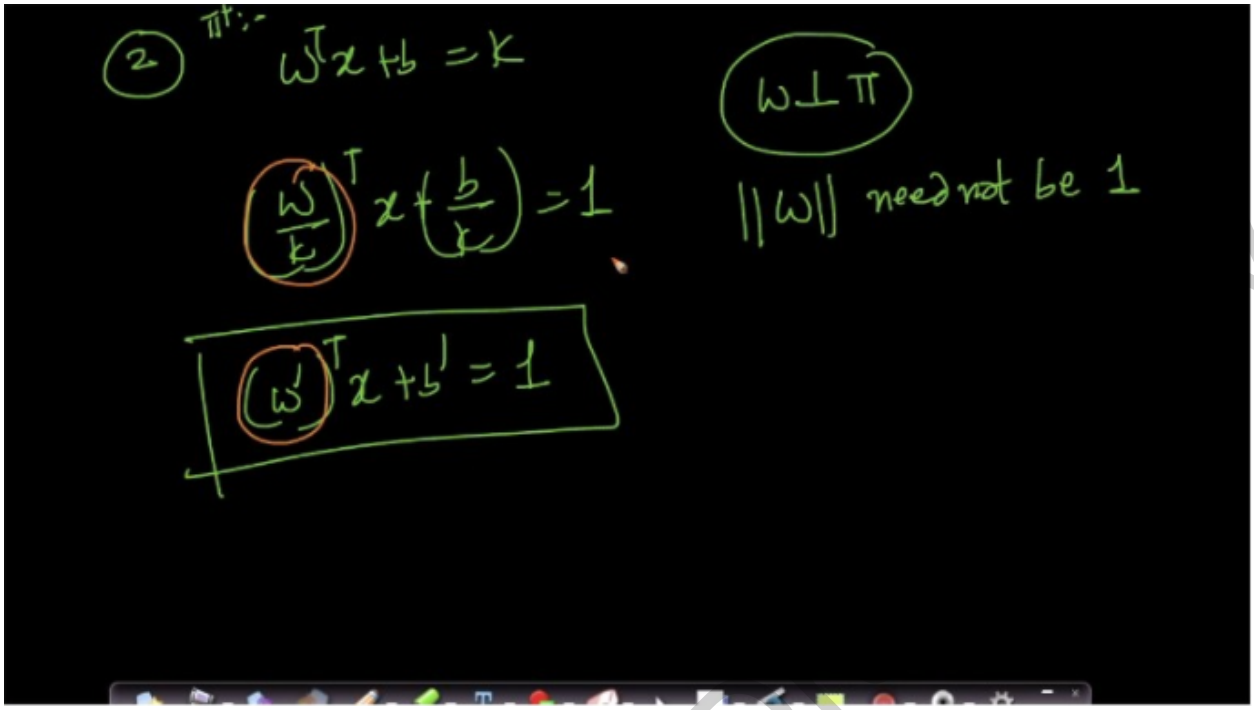
We will try to answer this question from multiple approaches.

1. Consider a case where we would have taken a constant k instead of 1 in the RHS. In that case our equations would have been

π_+ => $w^T x$ + b = k and π_- => $w^T x$ + b = -k.

Keep in mind that the hyperplane π needs to be in equal distance from both positive and negative hyperplanes, that's why we have taken the same RHS in both cases. k can take any positive value i.e. k>0 and it's a constant.
So, our margin becomes 2k / ||w||.
Then, our equation becomes
(w^*, b^*) = $argmax_{(w,b)}$ 2k / || w || , which is equivalent to
(w^*, b^*) = $argmax_{(w,b)}$ 2 / || w ||
because k is just a constant and both these equations will yield the same results.

So, for ease of calculation we have taken the value of k = 1.



Timestamp 6:47

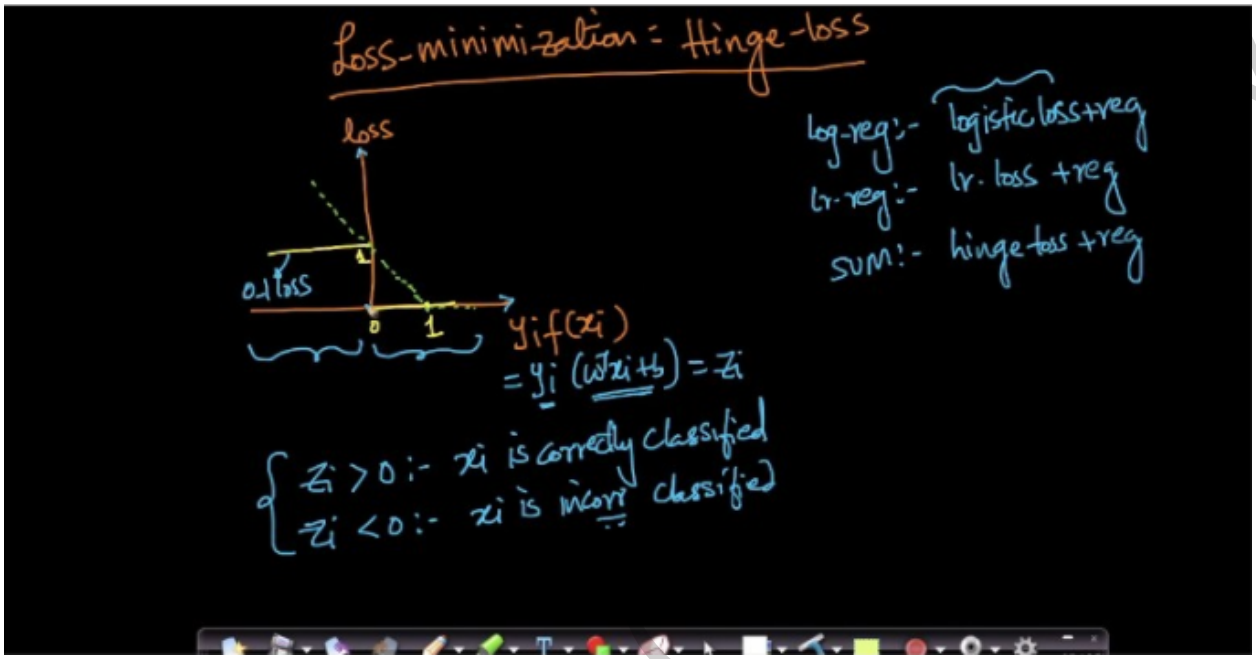
2. Say we taken our positive hyperplane as π_+ => $w^T x$ + b = k.

Now we can divide both sides by k, then, π_+ => $(w/k)^T x$ + (b/k) = 1.

We can rewrite this as π_+ => $(w^|)^T x + b^|$ = 1

Dividing w by k which is a constant won't change anything because our only requirement is that w should be perpendicular to π , which won't change if we divide it by a constant.

36.4 Loss function (Hinge Loss) based interpretation



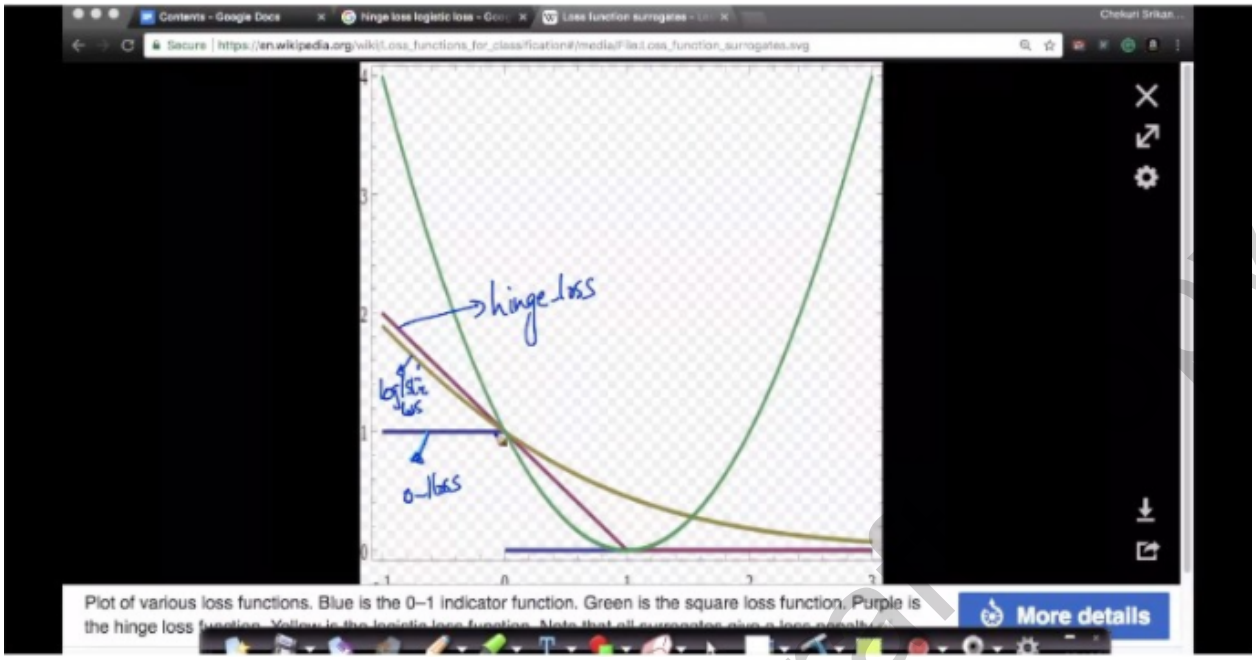
Timestamp 2:03

Here we will try to understand SVM from the perspective of hinge loss.
Remember when we studied logistic regression, it is logistic loss + regularizer. Similarly linear regression is linear loss + regularizer.
Similarly we saw that SVM is hinge loss + regularizer.

In the above image we can see a graph where on the y-axis we have loss and on the x-axis we have our prediction y_i* f(x_i). In case of SVM, f(x_i) = y_i* ($w^T x_i$ + b). Let's call this z_i.

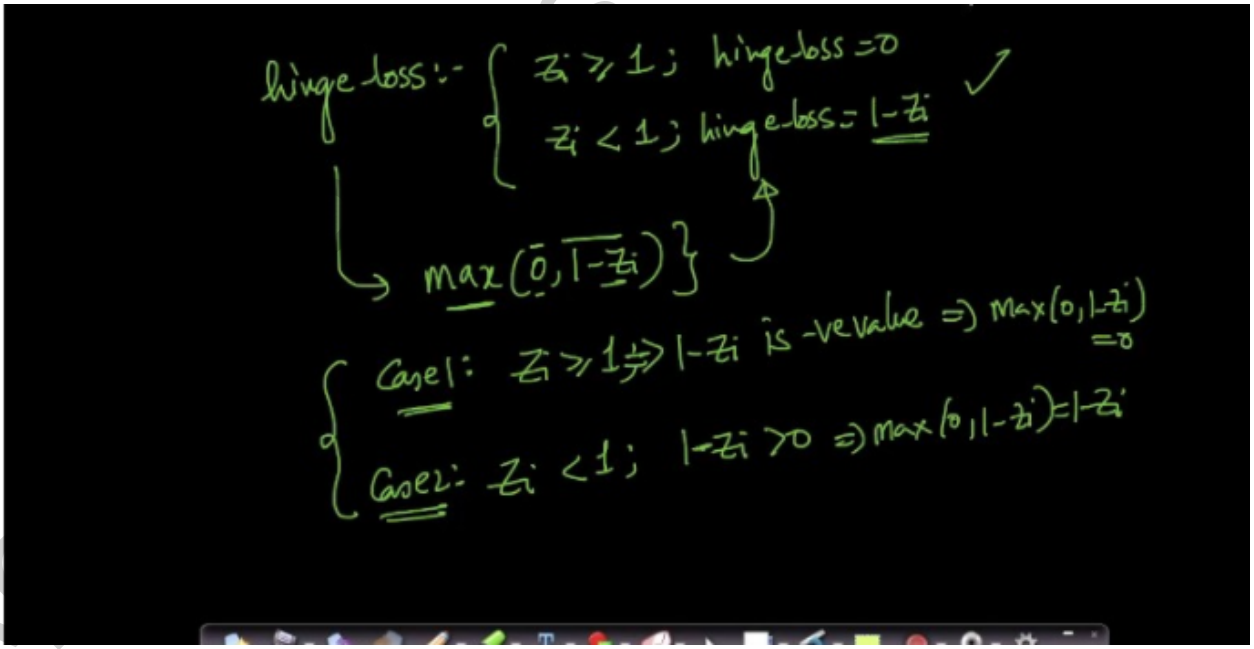
We can draw a 0 -1 loss for this as shown by the yellow curve on the graph. Here for 0 -1 loos we have loss = 1 for z_i < 0 and the loss = 0 for z_i > 0.

But as we know that 0-1 loss is not continuous at 0, hence it is not differentiable.
The green dotted line in the graph is the hinge loss.



Timestamp 2:47

We can see in the image above that the blue line represents the 0-1 loss. The brown line represents the logistic loss and the purple line represents the hinge loss. Both hinge loss and logistic loss are approximations of 0-1 loss.



Timestamp 7:08

Hinge loss is 0 if $z_i \geq 1$, and hinge loss is 1 - z_i if $z_i < 1$.

We can verify this using the graph that we have drawn earlier.

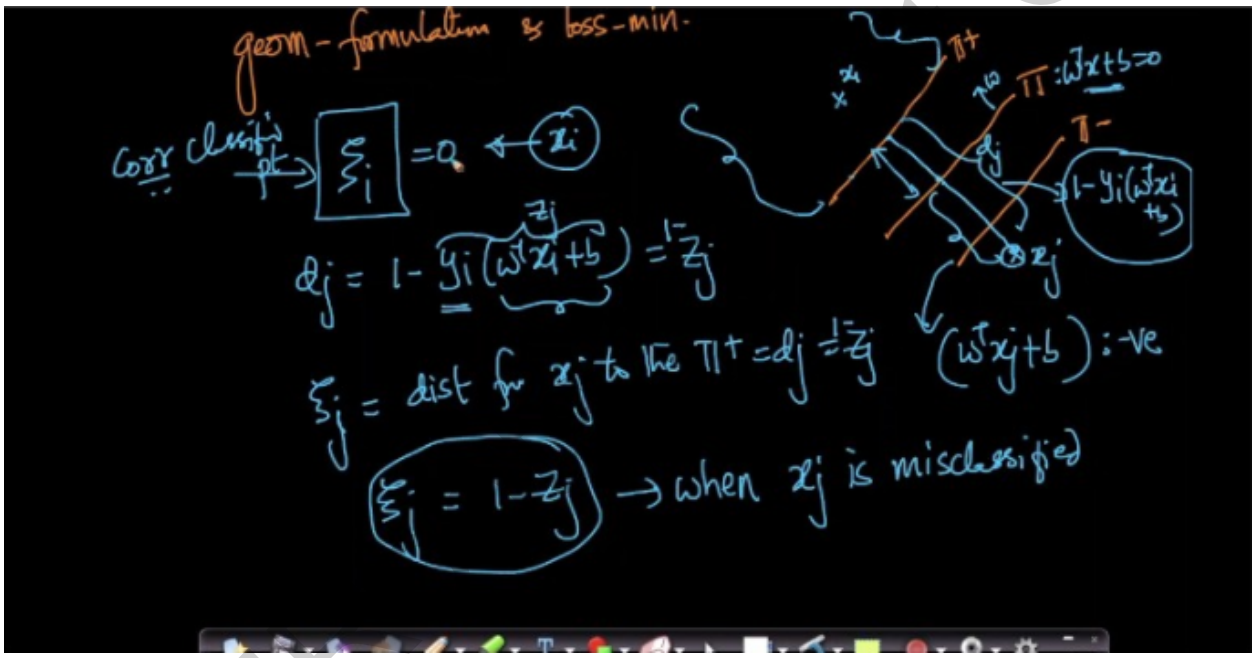
Hinge loss can also be written as => max(0 , 1 - z_i)

Let's check why this is true,
Case 1: if $z_i \geq 1$, then 1 - z_i is negative => max (0, 1 - z_i) = 0

Case 2 : if $z_i < 1$, then 1 - $z_i > 0$ => max(0, 1- z_i) =1 - z_i

These cases match our formulation of hinge loss. This is a more compact way to write hinge loss.

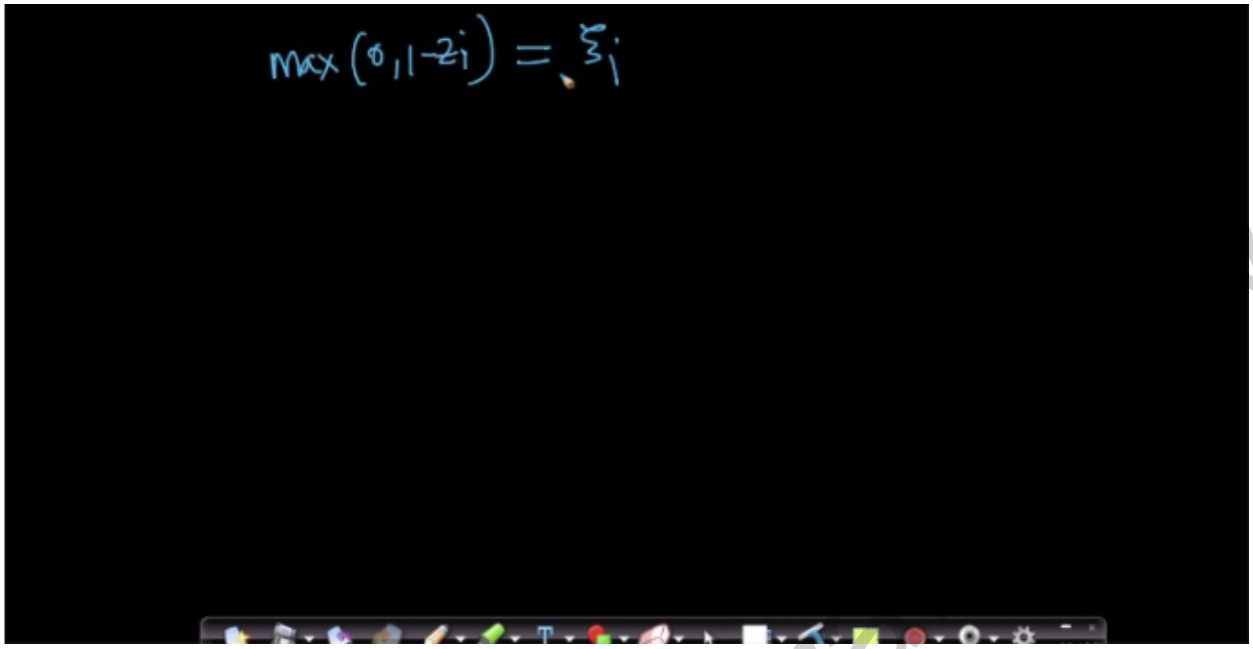
Also note that hinge loss is not differentiable, so in practice we use some hacks to get around this problem.



Timestamp 11:36

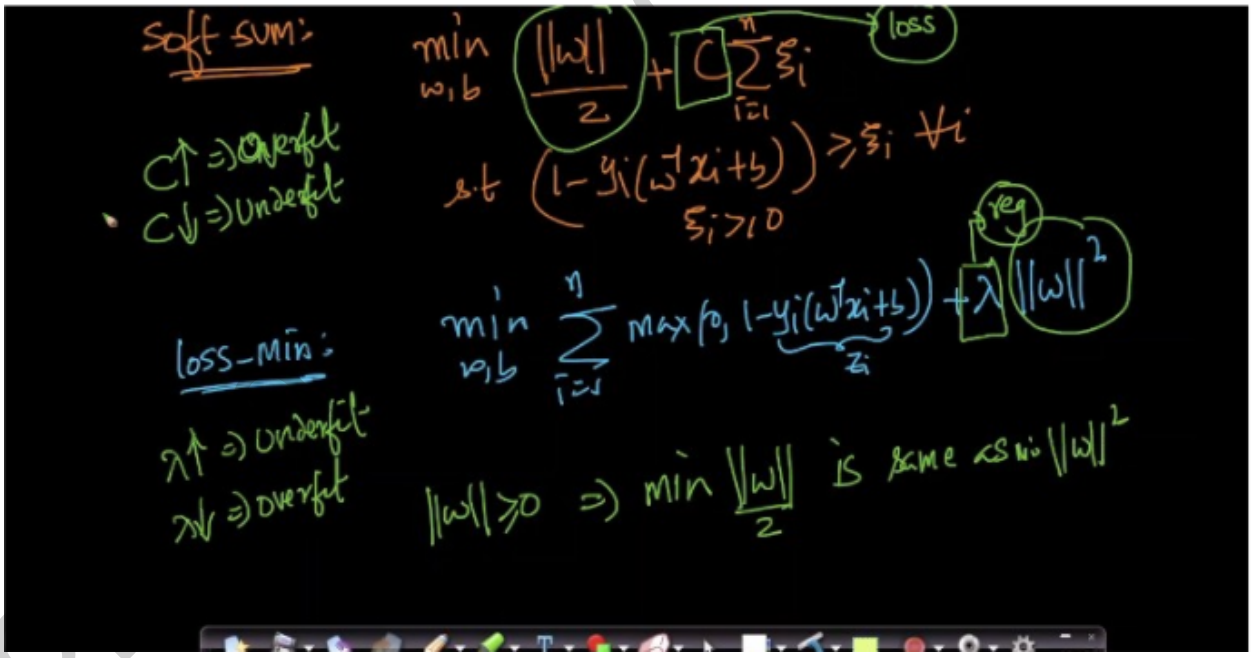
Let's try to connect the geometric formulation and loss minimization.

Consider a positive point x_i which is present on the correct side of the hyperplane. ξ_i for this data point will be 0.
Consider another positive data point x_j which is present on the side of the negative hyperplane.
The distance d_j of this point from the positive hyperplane is = 1 - y_j * ($w^T x_j$ + b) = 1 - z_j.
We also know that ξ_j for this point will represent the distance from the correct in our case positive hyperplane.
So, for misclassified points ξ_j = 1 - z_j



Timestamp 11:51

So we can see that max(0, 1 - z_i) = ξ_i . We saw the verification in our previous slide.



Timestamp 17:15

We have already seen that ξ_i is equivalent to max(0, 1 - z_i) under the constraints of ξ_i.

So, the loss minimization interpretation of soft margin SVM is

$(w^*, b^*) = argmin_{(w,b)} \sum_{i=1}^{n} \max(0, 1 - z_i) + \lambda||w||^2$

Here, $\sum_{i=1}^{n} \max(0, 1 - z_i)$ => loss part

$\lambda||w||^2$ => It's a regularizer with λ as the hyperparameter and $||w||^2$ is L_2 norm of w.

The above formulation is equivalent to the one we saw during geometric interpretation i.e.

$(w^*, b^*) = argmin_{(w,b)} || w || / 2 + C^* 1/n \sum_{i=1}^{n} \xi_i$

s.t. $\forall i$, $y_i^*(w^T x_i + b) \geq 1 - \xi_i$,

$\xi_i \geq 0$

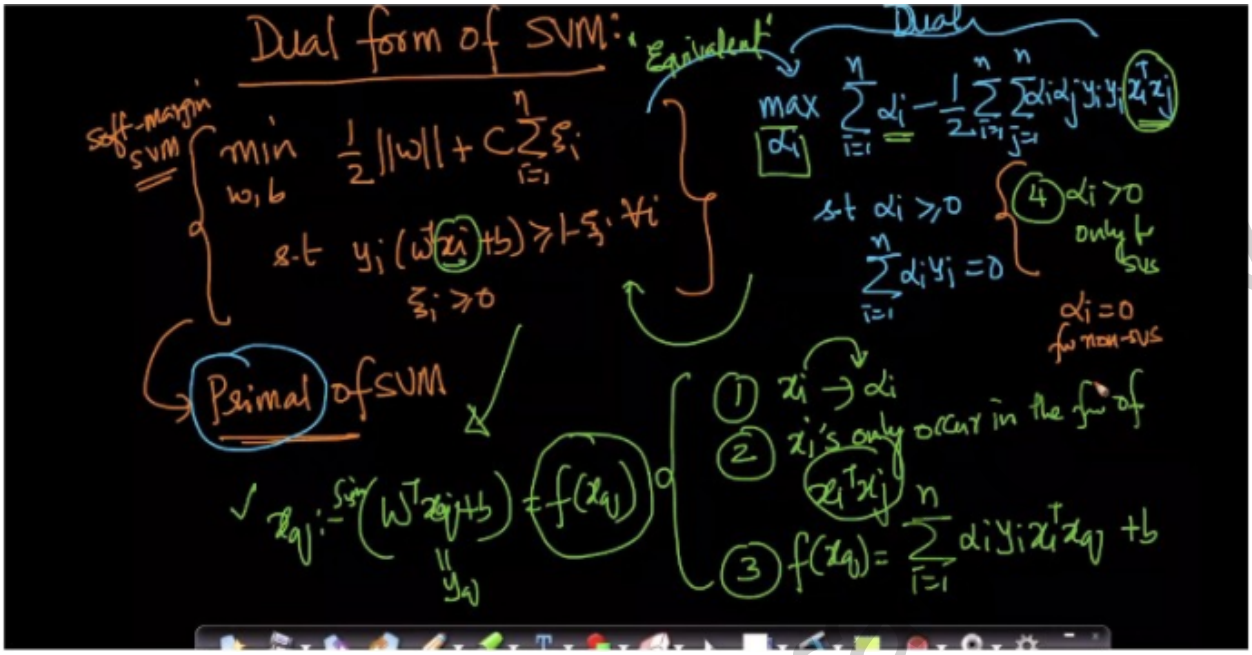
We have already seen that ξ_i is equivalent to $\max(0, 1 - z_i)$ [$z_i = y_i^*(w^T x_i + b)$]

Also it can be seen that we have changed || w || / 2 to $||w||^2$, because minimizing both of them are equivalent.

In the loss formulation the hyperparameter λ is multiplied to the regularizer, so if the λ increases, the model underfits and if it decreases, the model overfits.

In the geometric interpretation the hyperparameter C is multiplied to the loss, so if C increases, the model overfits, and if it decreases, the model underfits.

36.5 Dual form of SVM formulation



Timestamp 6:45

Here we will look at the dual formulation of SVM. We will take Soft Margin SVM here, because it is more useful in real life scenarios.

We earlier saw that the geometric formulation of Soft Margin SVM is

$(w^*, b^*) = argmin_{(w,b)} \| w \| / 2 + C^* 1/n \sum_{i=1}^{n} \xi_i$

s.t. $\forall i , y_i^*(w^T x_i + b) \geq 1 - \xi_i,$

$\xi_i \geq 0$

This is also called the Primal Form of SVM.

Now the Dual Form of SVM is

$(\alpha^*) = argmax_{(\alpha_i)} \sum_{i=1}^{n} \alpha_i - 1/2 \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j x_i^T x_j$

s.t. $C \geq \alpha_i \geq 0$

$\sum_{i=1}^{n} \alpha_i y_i = 0$

Using techniques from optimization it can be shown that both primal and dual forms are equivalent. The exact proof is beyond the scope of this course.

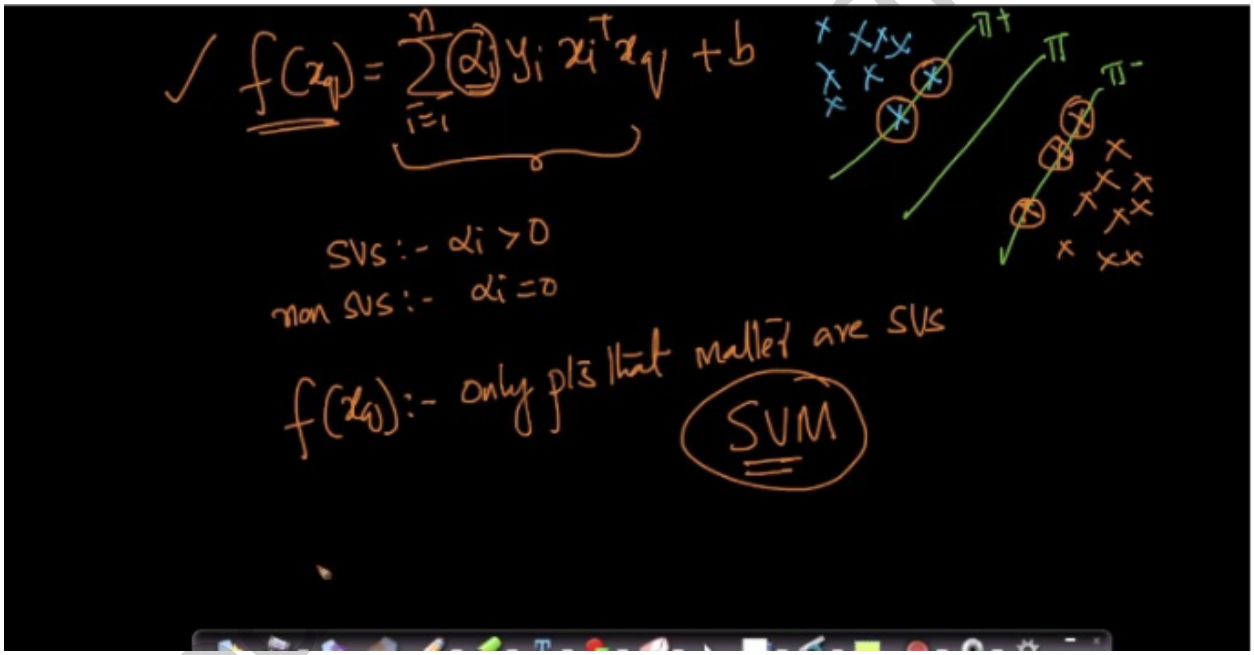
Let's see some properties of this dual form:

1. In dual form also we are summing over n i.e. for every $x_i \rightarrow \alpha_i$
2. x_i's only occur in the form of $x_i^T x_j$
3. In test time suppose we get a query point x_q, in the primal we can compute

f(x_q) = sign(($w^T x_q$ + b)). If the sign is positive we can call it a positive datapoint, otherwise a negative datapoint.
In the case of dual form we compute

f(x_q) =sign($\sum_{i=1}^{n} \alpha_i y_i x_i^T x_q$ + b)

4. α_i's are non-zero only for support vectors.



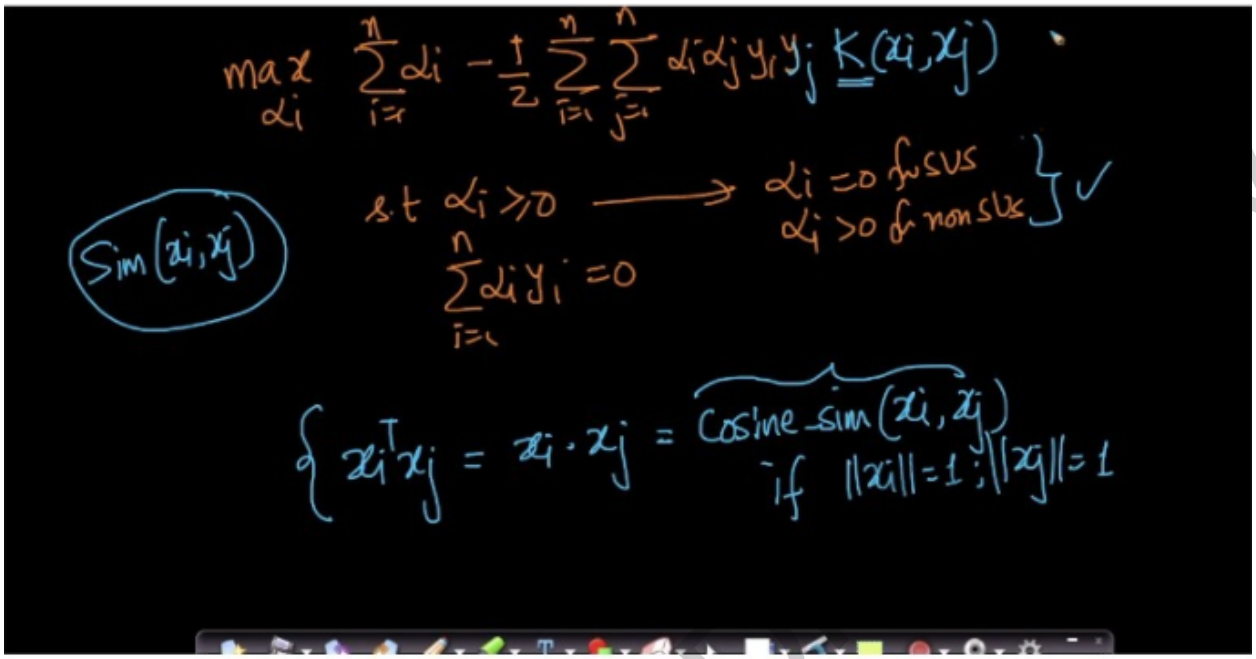
Timestamp 8:50

Let's look at the query function

f(x_q) = $\sum_{i=1}^{n} \alpha_i y_i x_i^T x_q$ + b

We can see from this function that we are iterating over all the α_i's, but as we know that α_i's are non-zero only for support vectors. So for the α_i's which are 0, the whole term for that becomes 0. So essentially we are summating over only the support vectors.

So for prediction only the support vectors matter and nothing else.



Timestamp 12:30

We also saw that x_i's only occur in the form of $x_i^T x_j$, here $x_i^T x_j$ is the dot product between x_i and x_j. It can be also thought of as the cosine similarity between x_i and x_j, if $||x_i|| = ||x_j|| = 1$.

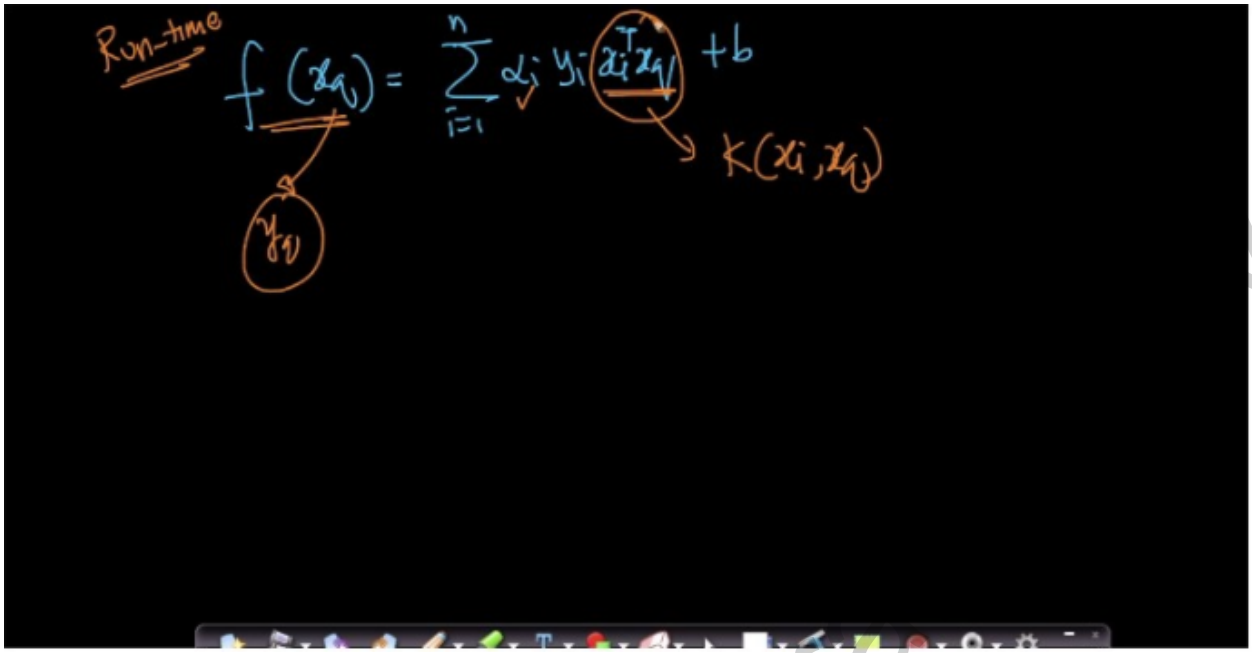
In place of cosine similarity we can use any similarity criteria sim(x_i , x_j). This is what makes SVM's very powerful.

Essentially we use something known as kernel functions(K), which we will see in later chapters. So we can write our dual form as:

$$(\alpha^*) = argmax_{(\alpha_i)} \sum_{i=1}^{n} \alpha_i - 1/2 \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(x_i, x_j)$$

s.t. $\alpha_i \geq 0$

$$\sum_{i=1}^{n} \alpha_i y_i = 0$$

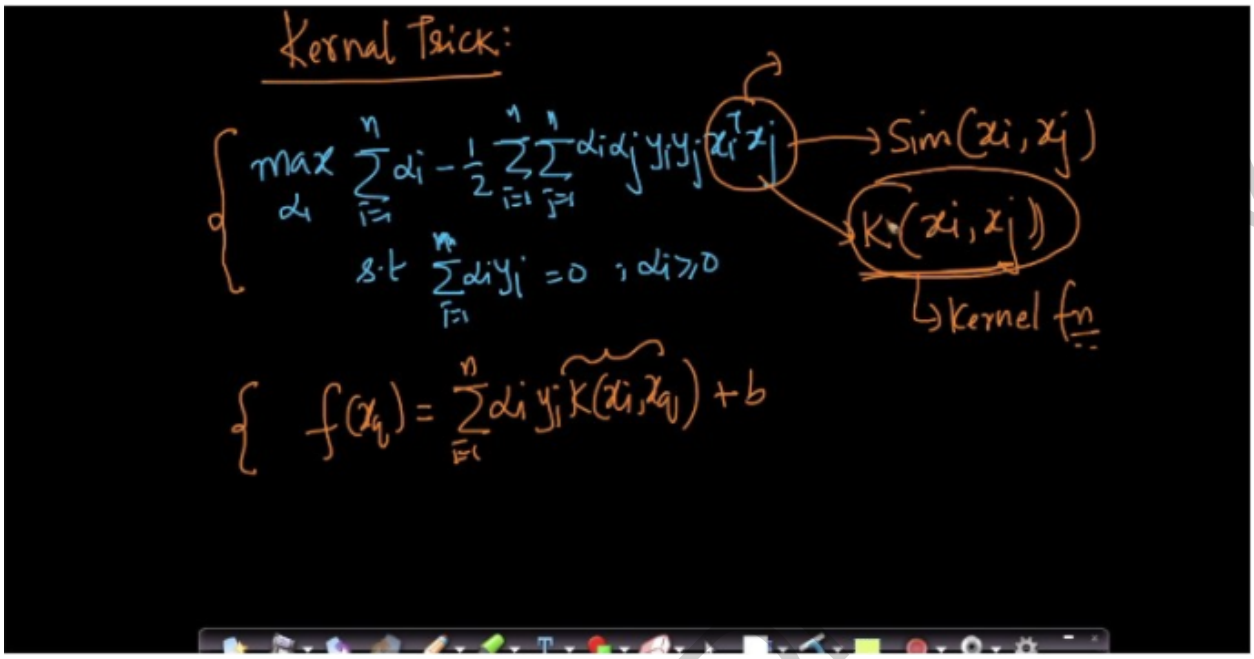


Timestamp 13:59

We can see that x_i's also occur in the form of $x_i^T x_j$ in the query function. So, here also we can use our kernel functions.

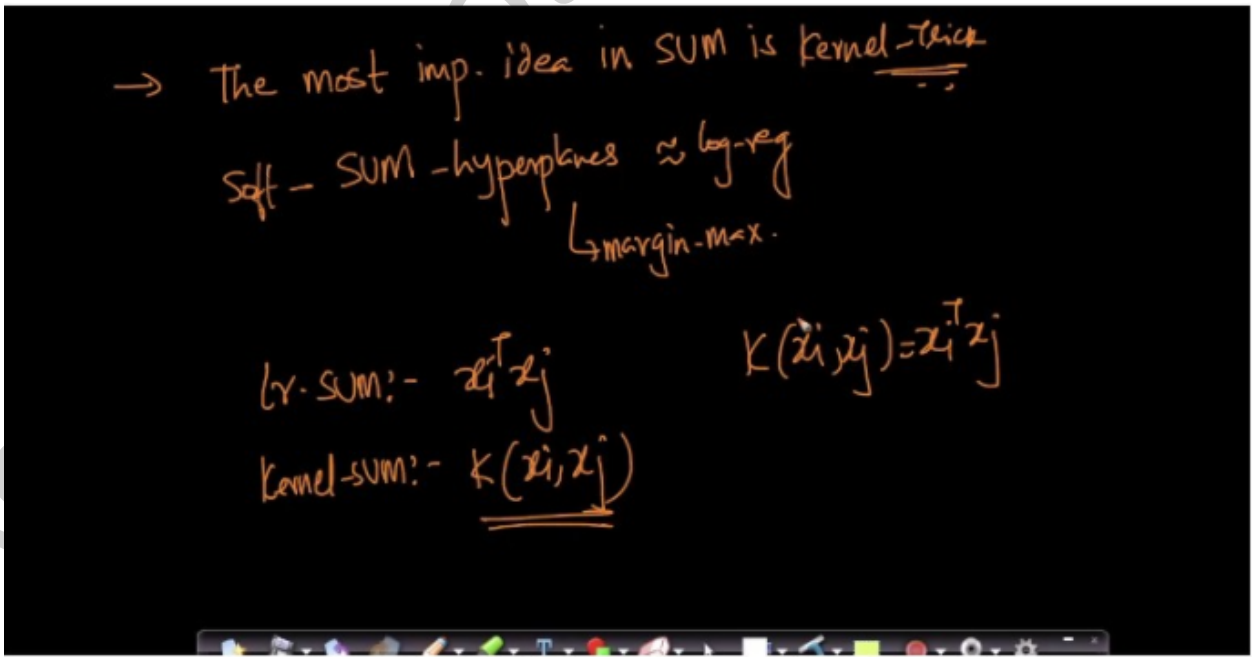
$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_q) + b$

36.6 kernel trick



Timestamp 2:02

As we saw in the previous chapter that instead of using $x_i^T x_j$, in our dual form we can use any similarity function. kernel functions are one special class of functions which can be used here. We can use the kernel function in the query function also, as we can see from the image above.

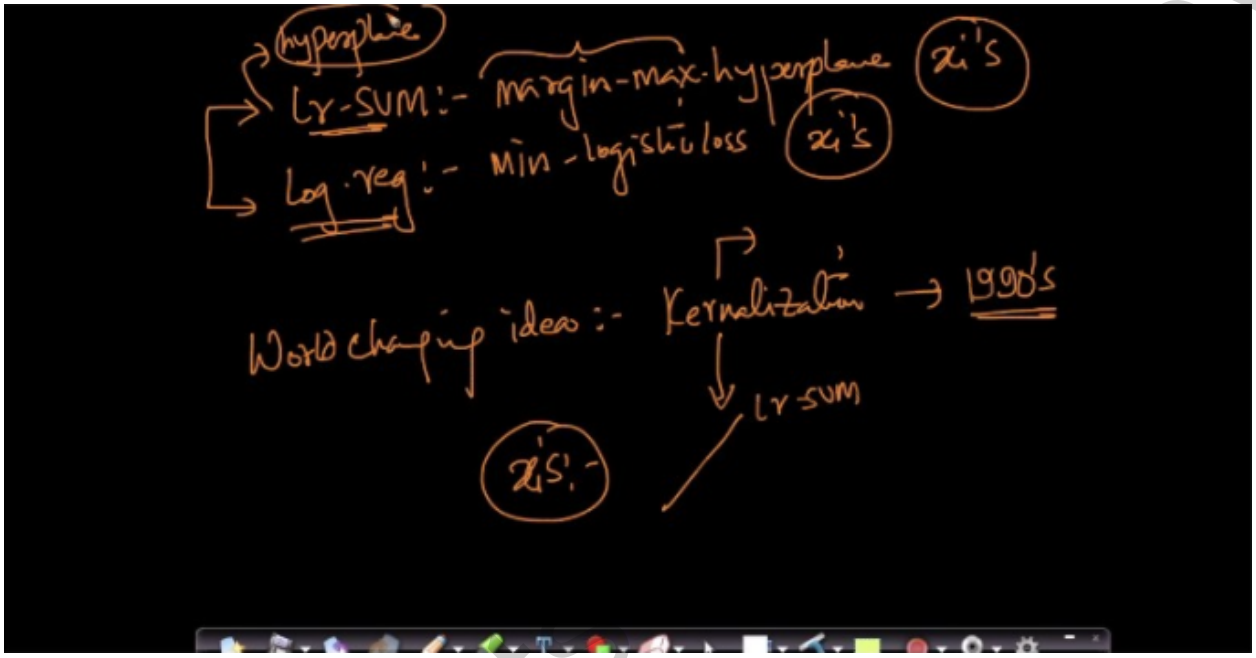


Timestamp 3:40

This is one of the most important ideas in SVM.

If we keep $x_i^T x_j$, as it is then the SVM is essentially known as linear SVM.

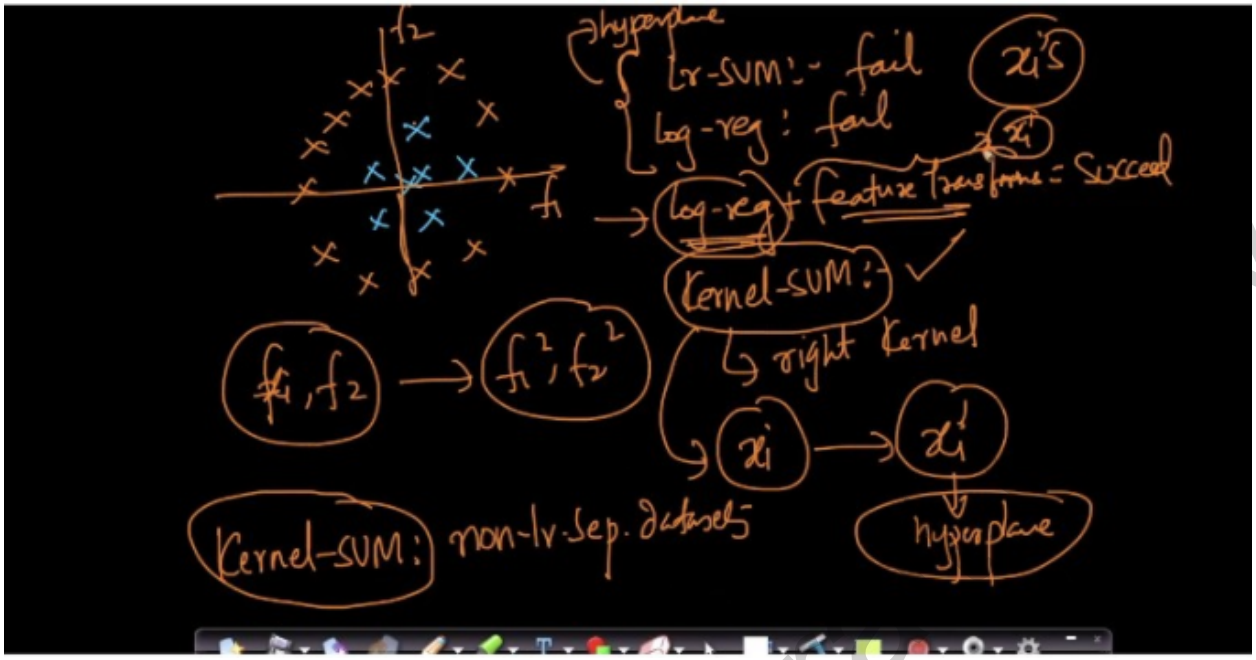
If we apply some kernel function at $x_i^T x_j$ i.e. K(x_i , x_j) then it is essentially known as kernel SVM.



Timestamp 5:40

So, essentially in linear SVM we are trying to find the margin maximizing hyperplane in the space of x_i's. Also in logistic regression we try to minimize the logistic loss in the space of x_i's.

So. linear SVM and logistic regression are quite similar and often produce similar results.

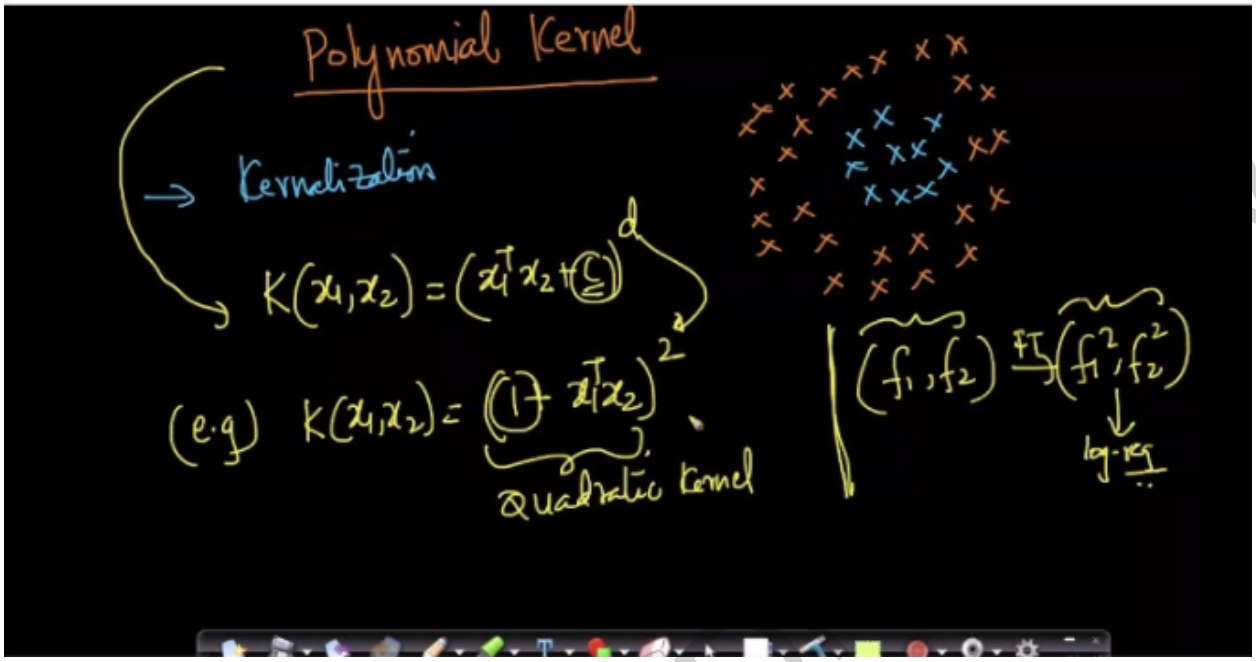


Timestamp 8:39

Consider a dataset as shown in the image above. Now for this dataset no linear hyperplane can work. So algorithm like linear SVM and logistic regression which work in the space of x_i's will fail in this case. Of course we can try feature engineering and transform these features into different spaces and see if the algorithm works.
So, linear SVM and logistic regression will fail in datasets which are not linearly separable.

Kernel SVM can work in this case, it tries to transform our features into some other higher dimensions such that we may find a hyperplane in this transformed space.

36.7 Polynomial Kernel



Timestamp 2:07

Here we will talk about the polynomial kernel, which is one of the types of kernel.
Consider a dataset as shown in the image above, it is basically two concentric circles. Now this is not linearly separable, so in order to fit a hyperplane we can perform a feature transformation say our features f_1 and f_2 becomes f_1^2 and f_2^2 and then use logistic regression.

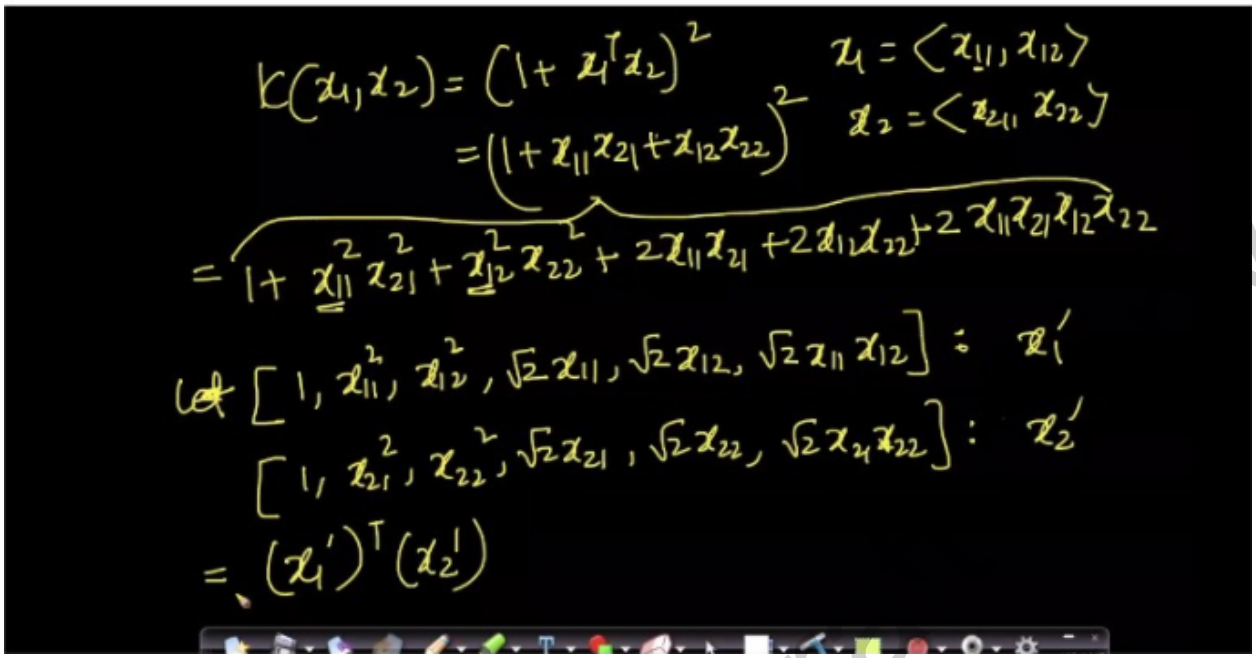
Now let's see how Kernelization solves this problem.
Polynomial Kernel :

$K(x_1, x_2) = (x_1^T x_2 + c)^d$

where c is a constant and d is the power of the polynomial.

Consider a case of quadratic kernel in this case c = 1, d = 2

So, $K(x_1, x_2) = (x_1^T x_2 + 1)^2$



Timestamp 5:17

Suppose x_1, x_2 are 2D vectors, where x_1 =< x_{11}, x_{12}> and x_2 =< x_{21}, x_{22}>

So, K(x_1, x_2) = $(1 + x_{11}x_{21} + x_{12}x_{22})^2$

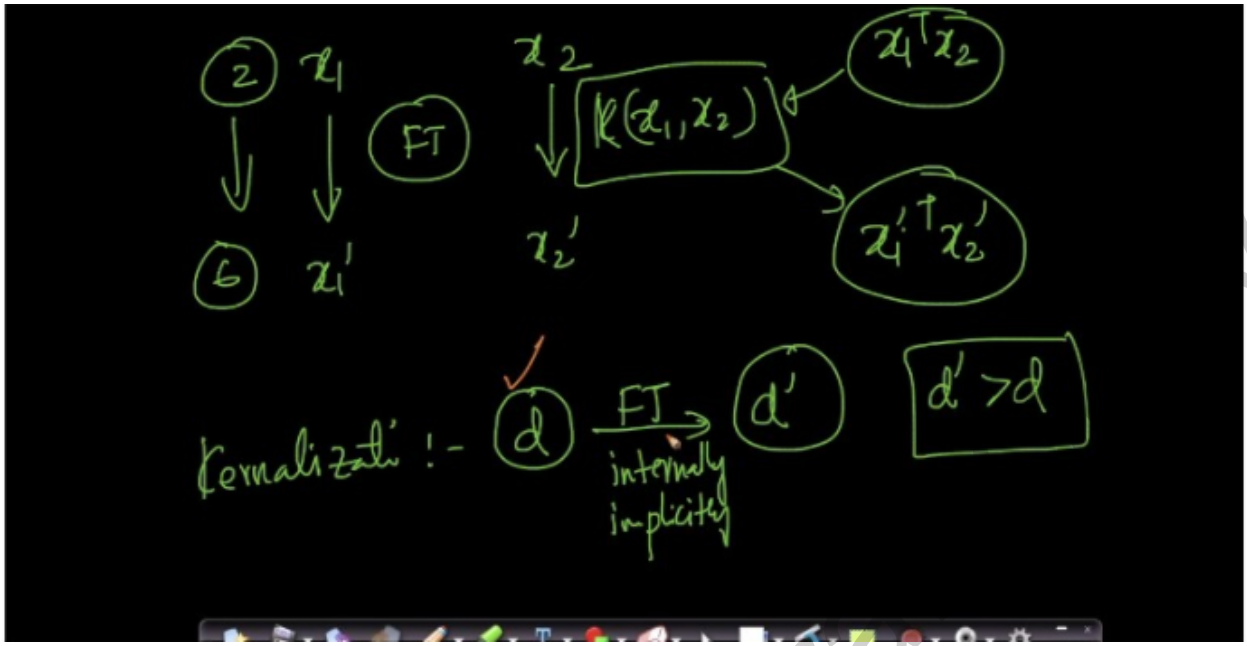
= 1 + $x_{11}^2 x_{21}^2$ + $x_{12}^2 x_{22}^2$ + $2x_{11}x_{21}$ + $2x_{12}x_{22}$ + 2 $x_{11}x_{21}x_{12}x_{22}$

This above term can be term be written as product of two vectors say x_1' and x_2', where

x_1' = [1, x_{11}^2, x_{12}^2, $\sqrt{2}x_{11}$, $\sqrt{2}x_{12}$, $\sqrt{2}x_{11}$ x_{12}]

x_2' = [1, x_{21}^2, x_{22}^2, $\sqrt{2}x_{21}$, $\sqrt{2}x_{22}$, $\sqrt{2}x_{21}$ x_{22}]

So, K(x_1, x_2) = $(x_1')^T$ (x_2')

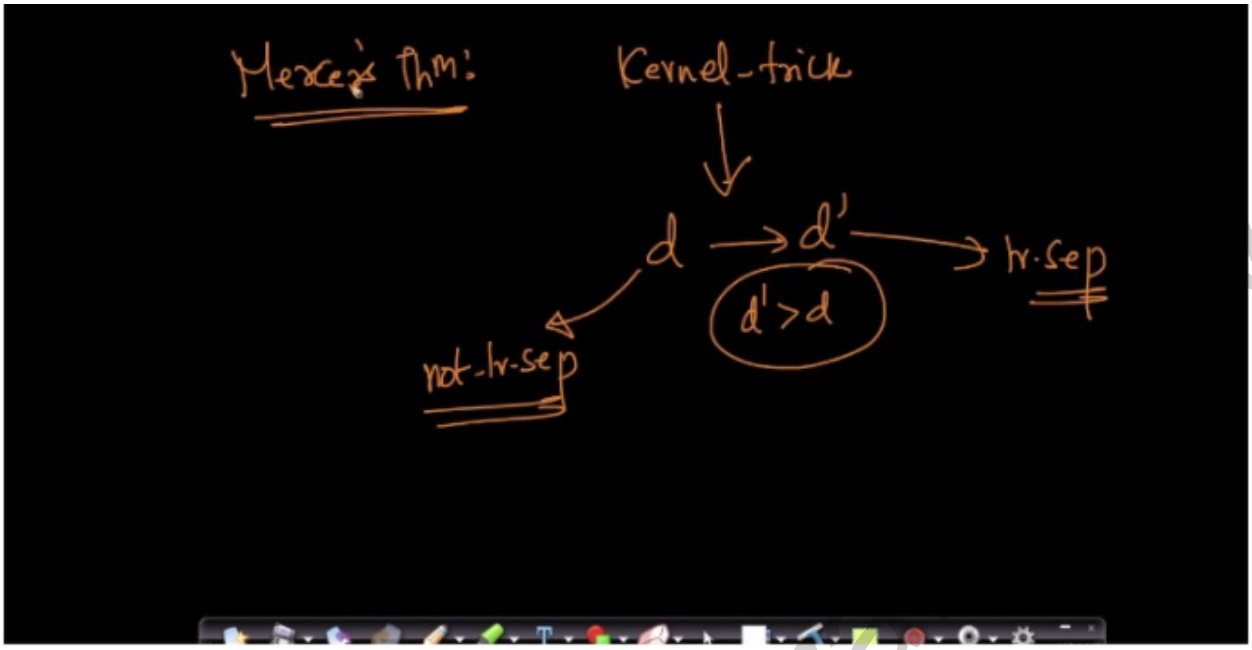


Timestamp 8:00

So essentially what we did was to take our data point which was in 2D and converted it into 6D as $x_1^{|}$ and $x_2^{|}$ contains 6 terms each.

Kernelization transformed our data from d dimensions into $d^{|}$, where usually $d^{|}$ > d. So kernelization is just like feature transformation where we transformed our features. In kernelization we did it implicitly/internally unlike logistic regression where we did it explicitly. This is essentially known as the kernel trick.

In our example we took dot product of transformed features K(x_1, x_2) = $(x_1^{|})^T (x_2^{|})$. These features took our data point into higher dimensions. It contained a squared term similar to feature transformation that we did in logistic regression, so we will be able to find a hyperplane here.



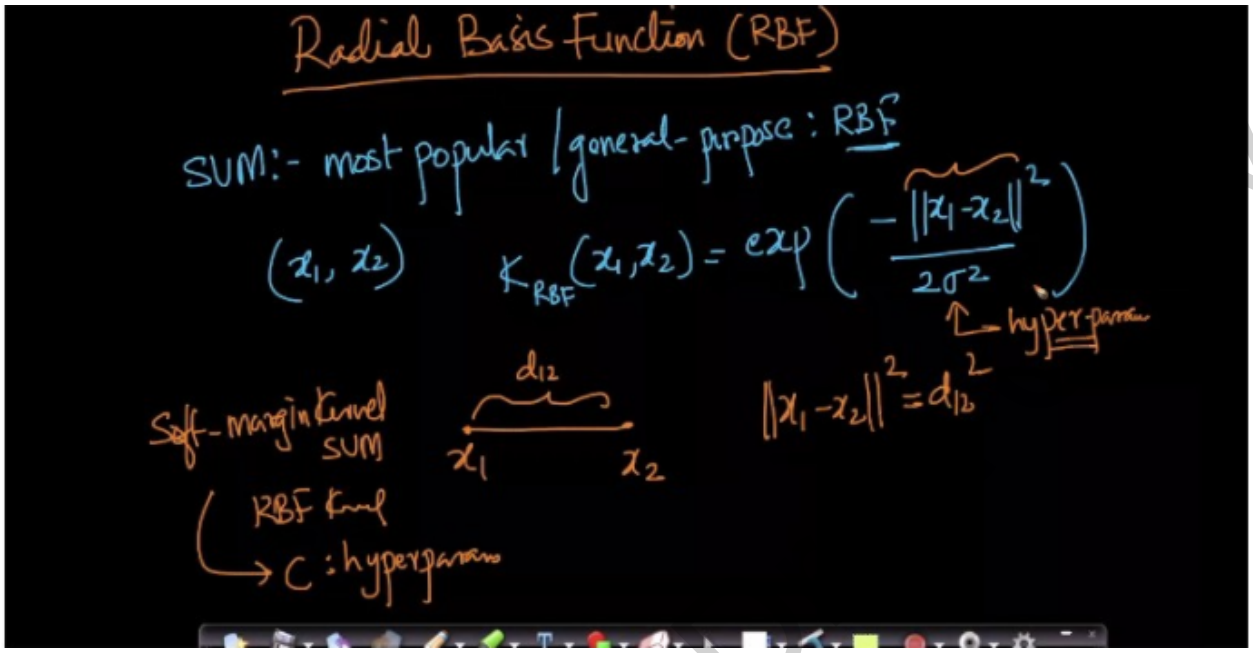
Timestamp 9:00

Mercer's theorem : It essentially says if we are converting our data points using kernelization or kernel trick from d dimension to $d^{|}$, where $d^{|}$ > d, it makes the data points linearly separable which was not the case in lower dimensions.

There is of course a proof behind this, but we won't go into much detail.

Now the problem is to find the right kernel. We saw in our concentric circle dataset that the polynomial kernel worked just fine. But it may not be the case in other datasets.

36.7 RBF-Kernel



Timestamp 1:15

We will now look at one of the most general purpose kernels.

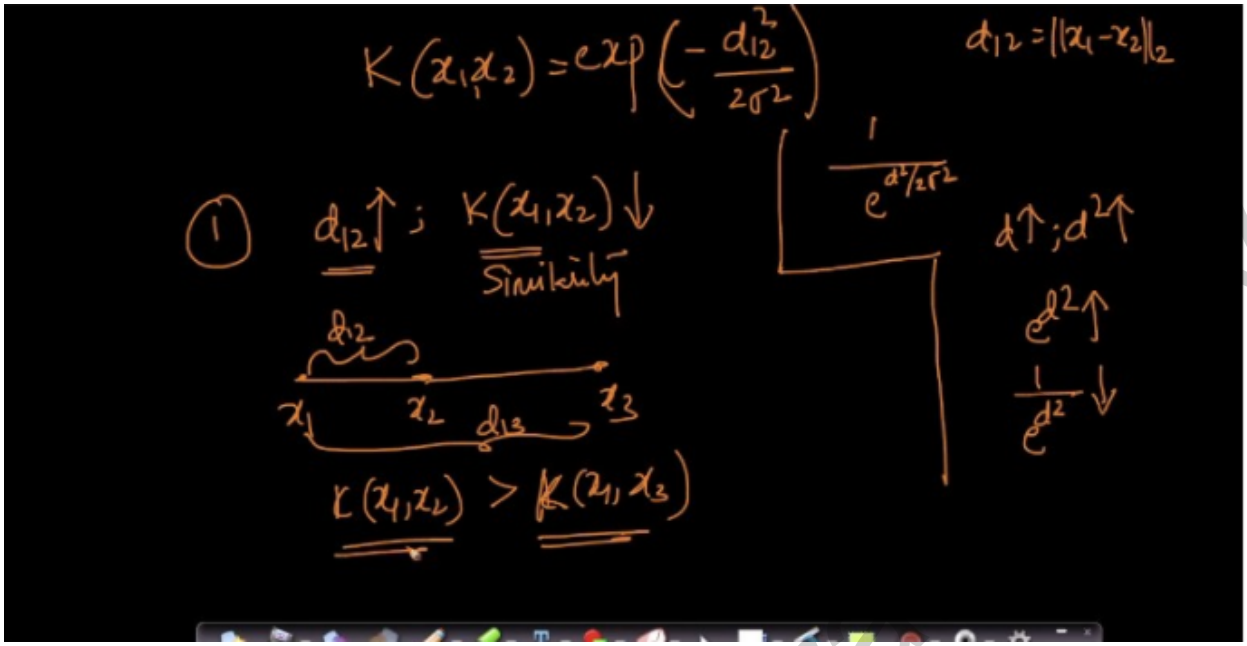
RBF or Radial Basis Function kernel between two data points x_1, x_2 is defined as :

$K_{RBF}(x_1, x_2) = \exp(- || x_1 - x_2 ||^2 / 2\sigma^2)$

where $|| x_1 - x_2 ||^2$ is the distance between two data points x_1 and x_2 ,

σ is a hyperparameter

$|| x_1 - x_2 ||$ can also be written as d_{12}, representing the distance.

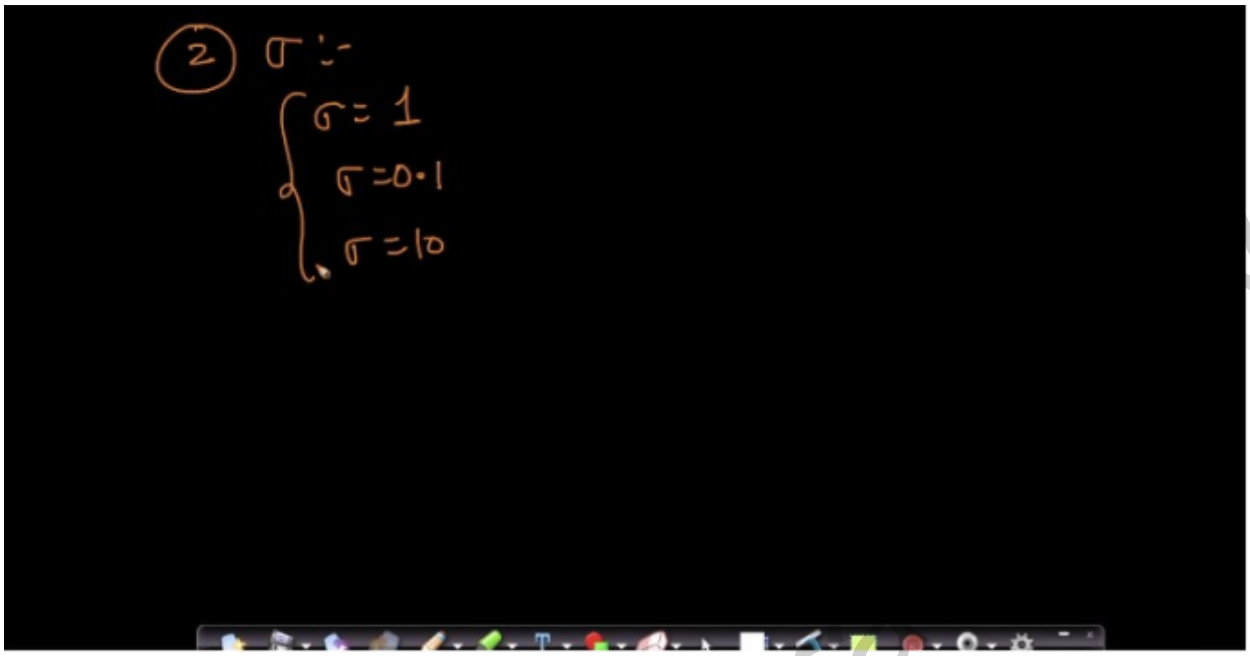


Timestamp 4:10

Now let's try to understand how this function behaves.

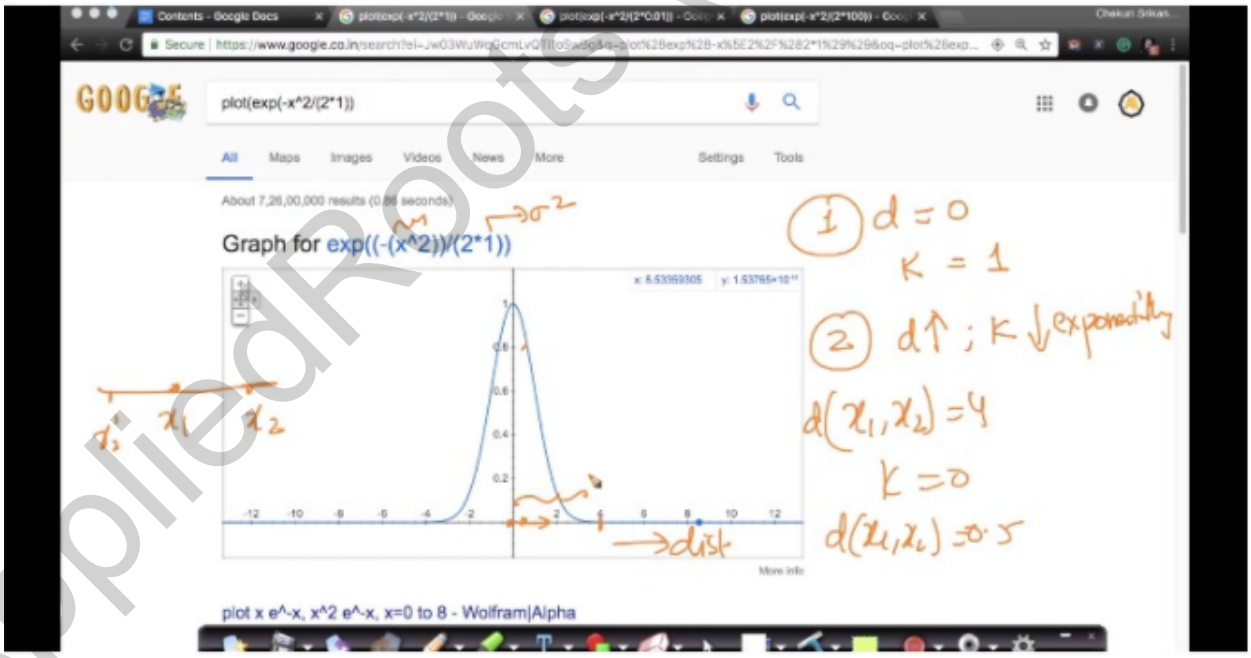
$K_{RBF}(x_1, x_2)$ = exp(- d_{12}^2 / $2\sigma^2$)

1. If the value of d_{12}^2 increases the value of $K_{RBF}(x_1, x_2)$ decreases as it can be seen from the image above. We know that d_{12} signifies the distance between x_1, x_2. So, $K_{RBF}(x_1, x_2)$ can be thought of as some sort of similarity where increase in distance is reducing the kernel value.



Timestamp 4:25

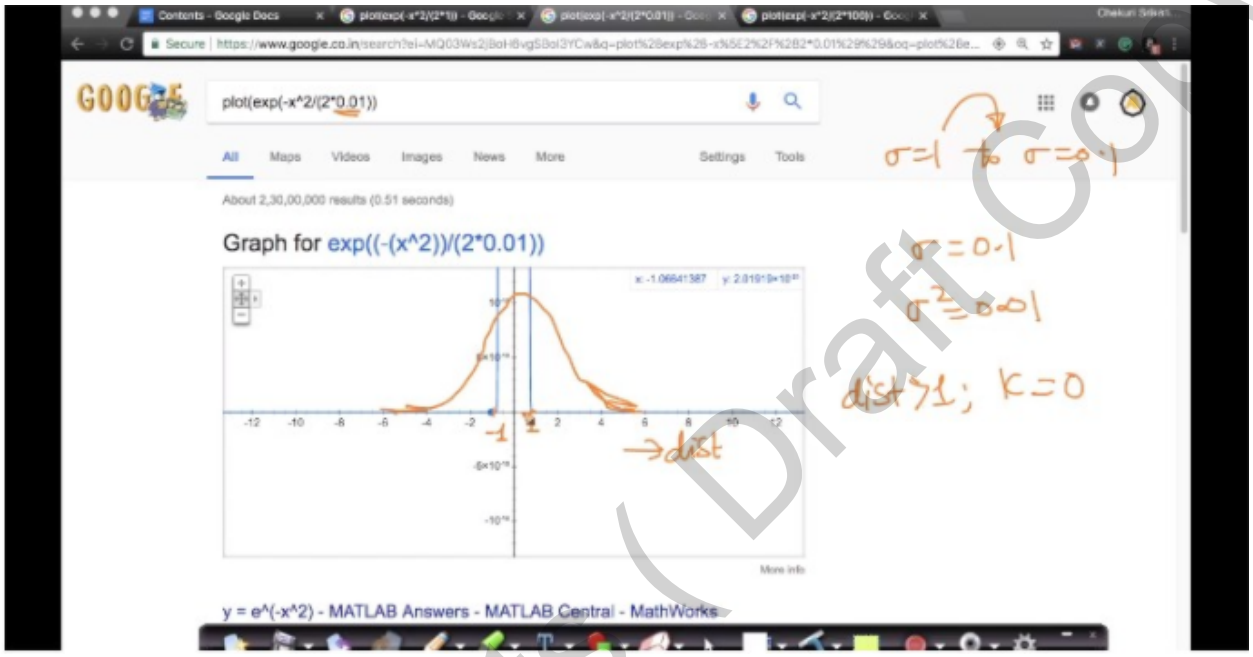
2. Let's try to see the impact of σ, for that we will see plots of $K_{RBF}(x_1, x_2)$ for different values of σ . Say we take 1, 0.1, 10



Timestamp 7:01

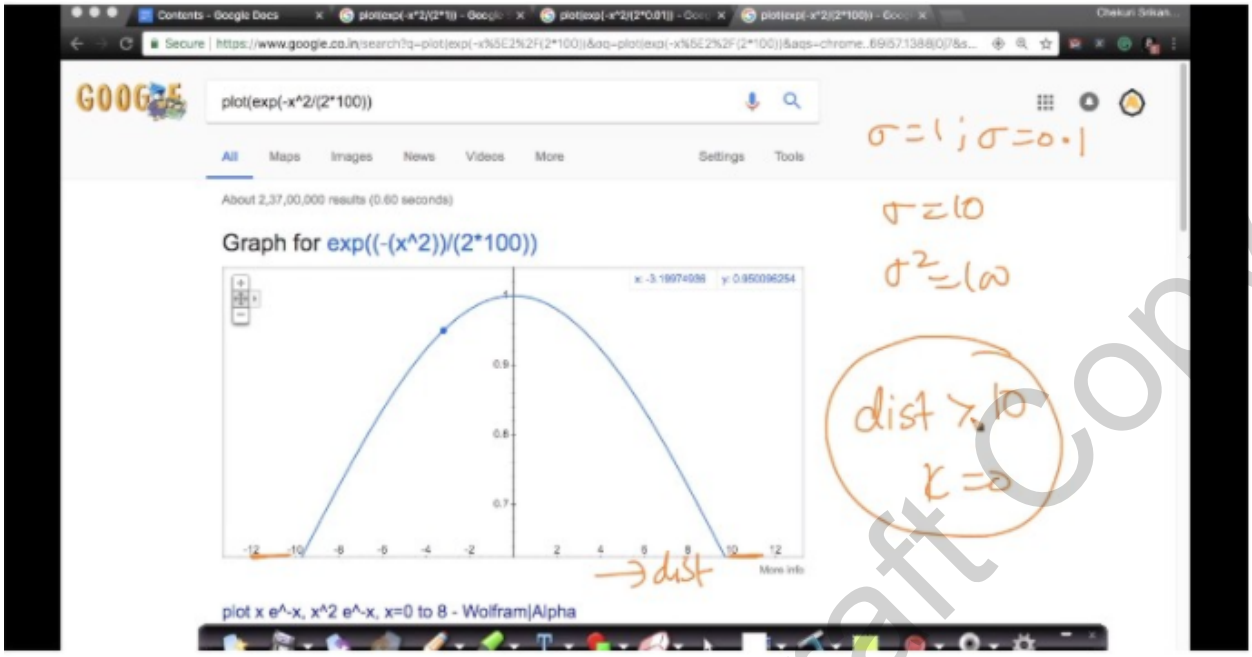
In the above plot we have drawn the RBF kernel with σ = 1. In the x-axis we represent the distance between two points and in the y-axis we have taken the rbf-kernel. We can see from

this plot that if the distance between two points is 0, the RBF kernel has a maximum value 1. This simply means that points which are at 0 distance away from each other are the most similar. Similarly we can see that as the distance increases, the rbf-kernel values fall exponentially. We can see that around distance 4 the rbf-kernel has a value of 0. Also note that the plot is symmetric since we are dealing with the squared distance. The plot looks like a gaussian curve because the formula is quite similar.



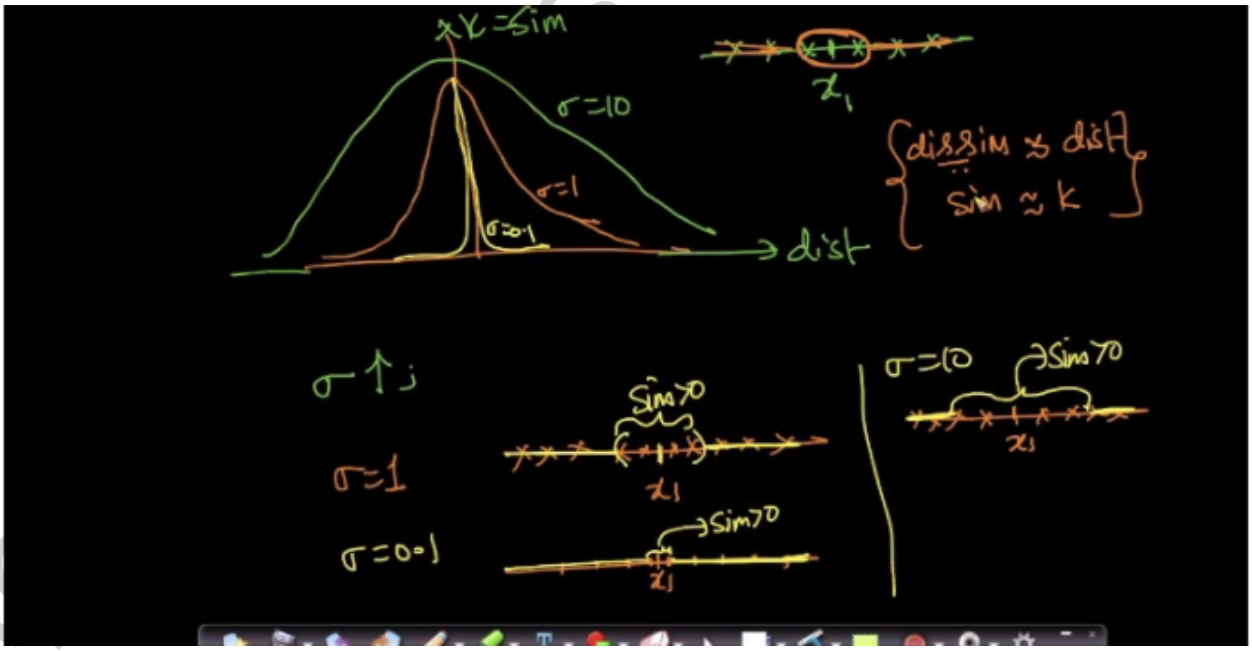
Timestamp 10:40

In the above plot we have drawn the rbf kernel with $\sigma = 0.1$. We can see that the curve has fallen faster than it did when $\sigma = 1$. For distances greater than 1, the rbf kernel has a value 0. The curve is much more peaked than the earlier curve.



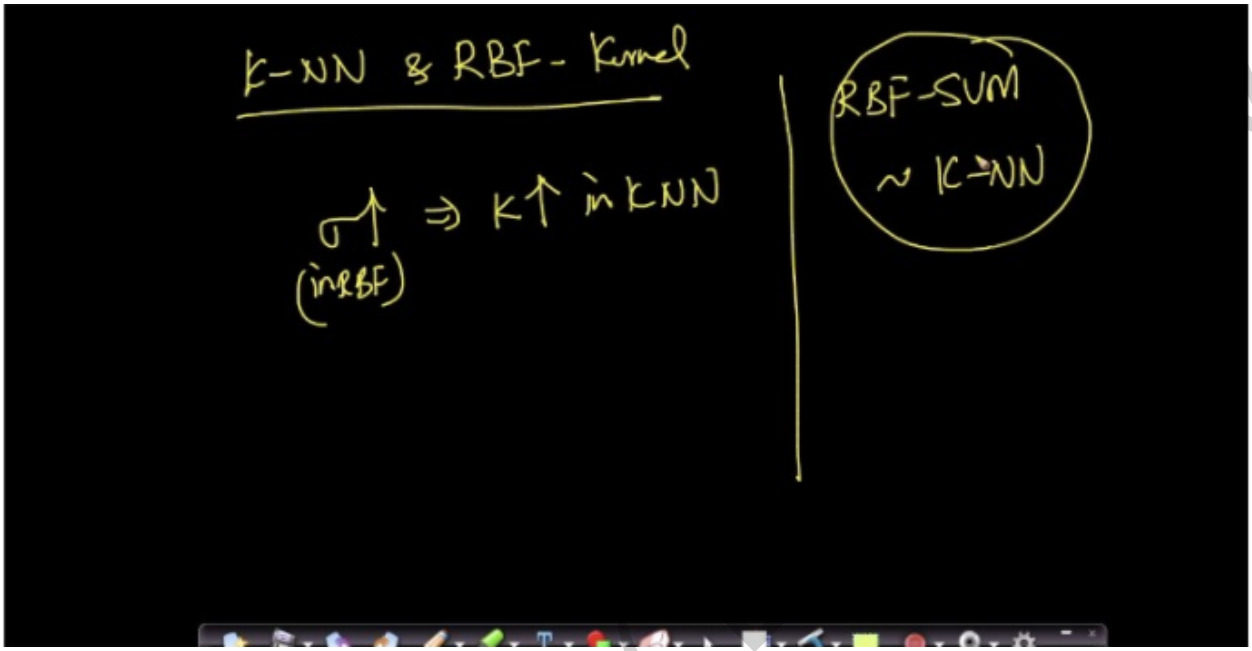
Timestamp 11:33

Now let's increase the value of σ. In the above plot we have drawn the rbf kernel with σ = 10. We can see that the curve is much wider than the earlier ones. For distances greater than 10, the rbf kernel has a value 0.



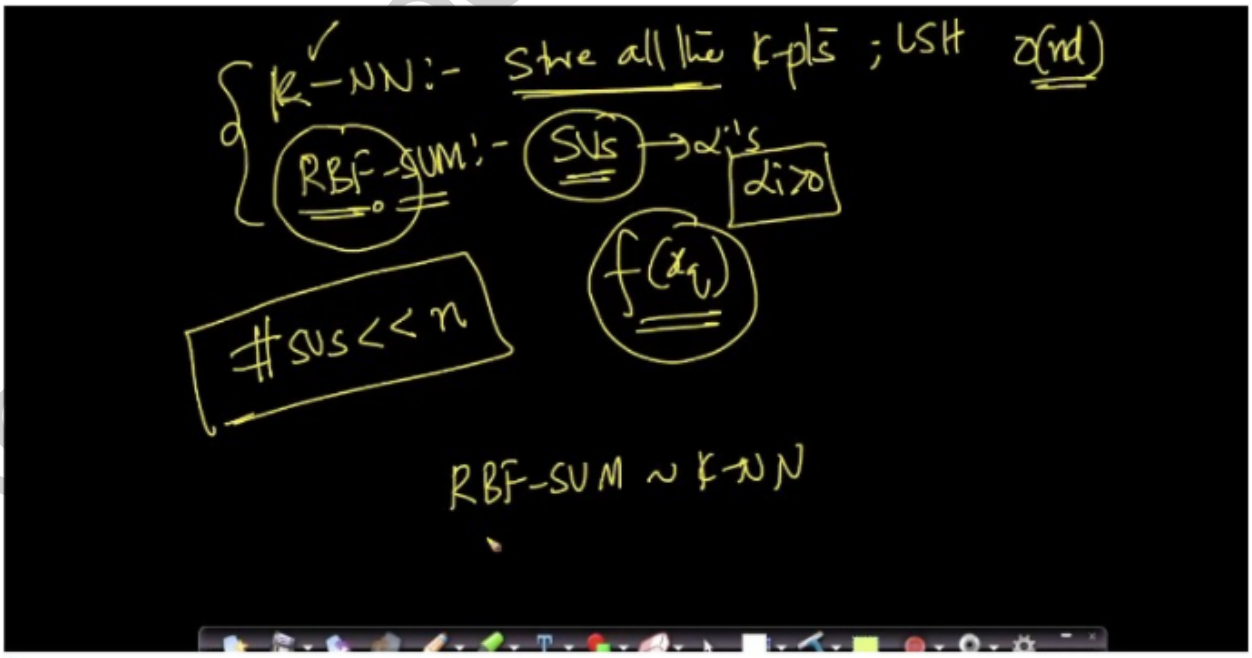
Timestamp 15:20

In the above plot we have drawn all the plots for different σvalues. We can see that as the value of σ is increasing we are allowing more points having larger distance to have a similarity value > 0. We can verify this from the above image.



Timestamp 17:24

There is a striking similarity between the rbf kernel and KNN. As we saw earlier that as the value of σincreased we had more points with a large distance having a similarity value. This is similar to K in KNN where a larger K means we can have more number of neighbours.

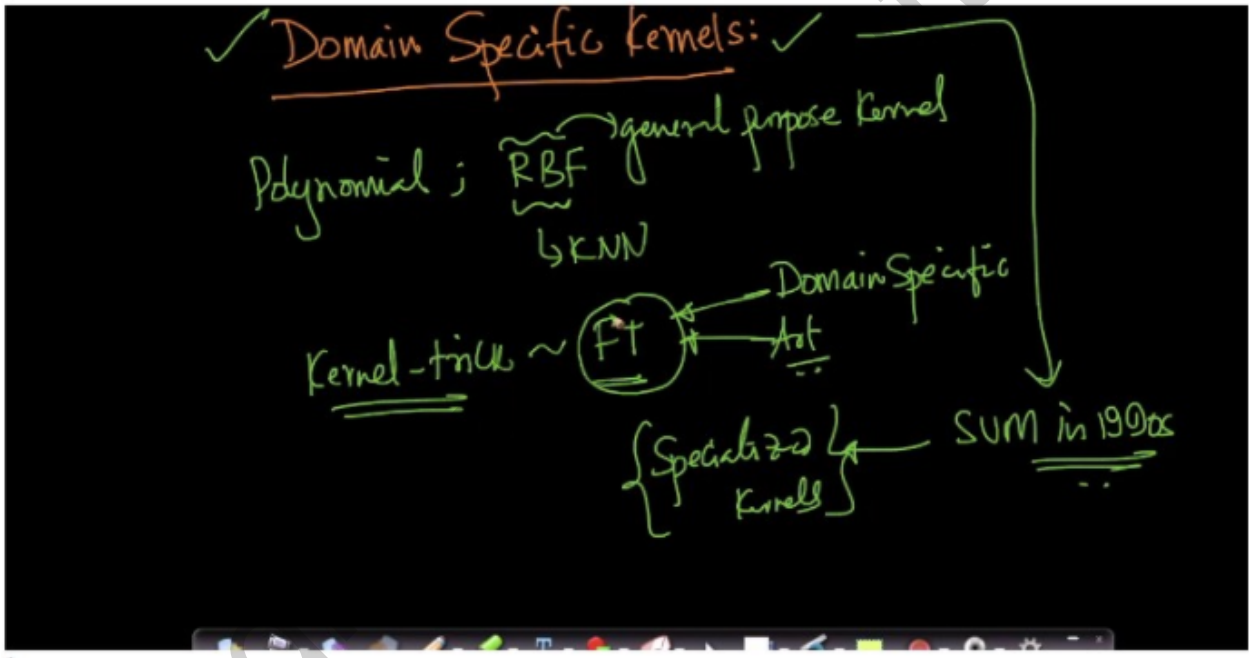


Timestamp 19:22

We know that in the case of KNN at test time we need all the data points in the memory, but for rbf kernels based SVM we only need support vectors. Generally #support vectors ≪#datapoints. So the rbf kernel kernel is a nice approximation to KNN. This is the reason why rbf kernels are the best general purpose kernels.

If we don't know what kernel to use we can always go for the rbf kernel. There is however one drawback that now we need to find two hyperparameters i.e. C and σ.

36.8 Domain specific Kernels

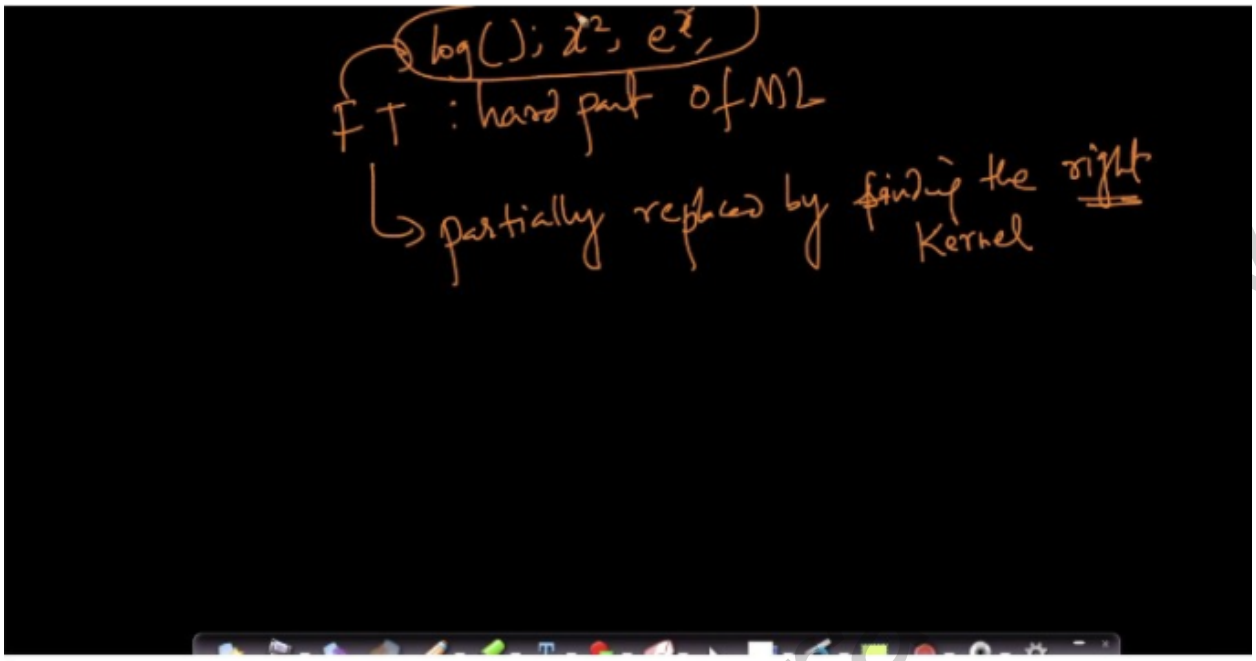


Timestamp 1:25

Till now we have seen a lot of kernels like polynomials, rbf kernels , etc. We have also seen that the kernelization is essentially the same as the feature transformation. Remember feature transformation is very domain specific. It is an art rather than science.

In the 1990's many specialized kernels were invented to suit various domains. Like there are string kernels, genome kernels, graph based kernels .etc.

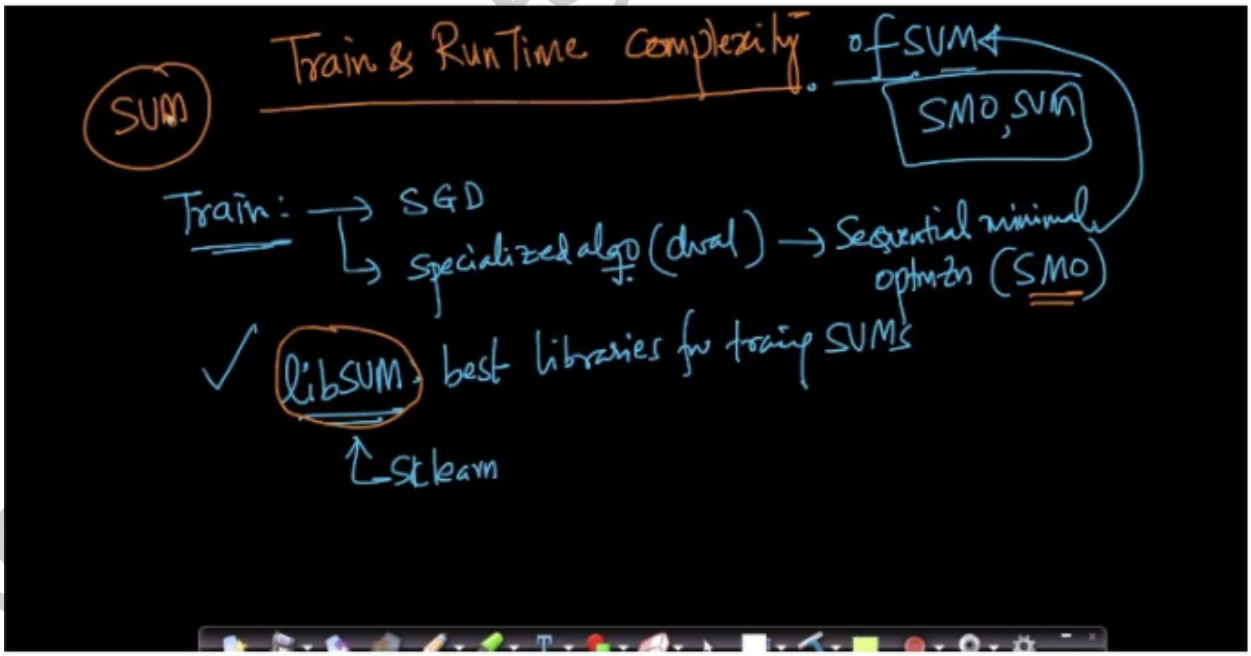
The choice of kernel depends on the type of problem that we are trying to solve.



Timestamp 5:23

So effectively feature transformation is replaced partially by finding the right kernel.

36.10 Train and run time complexities

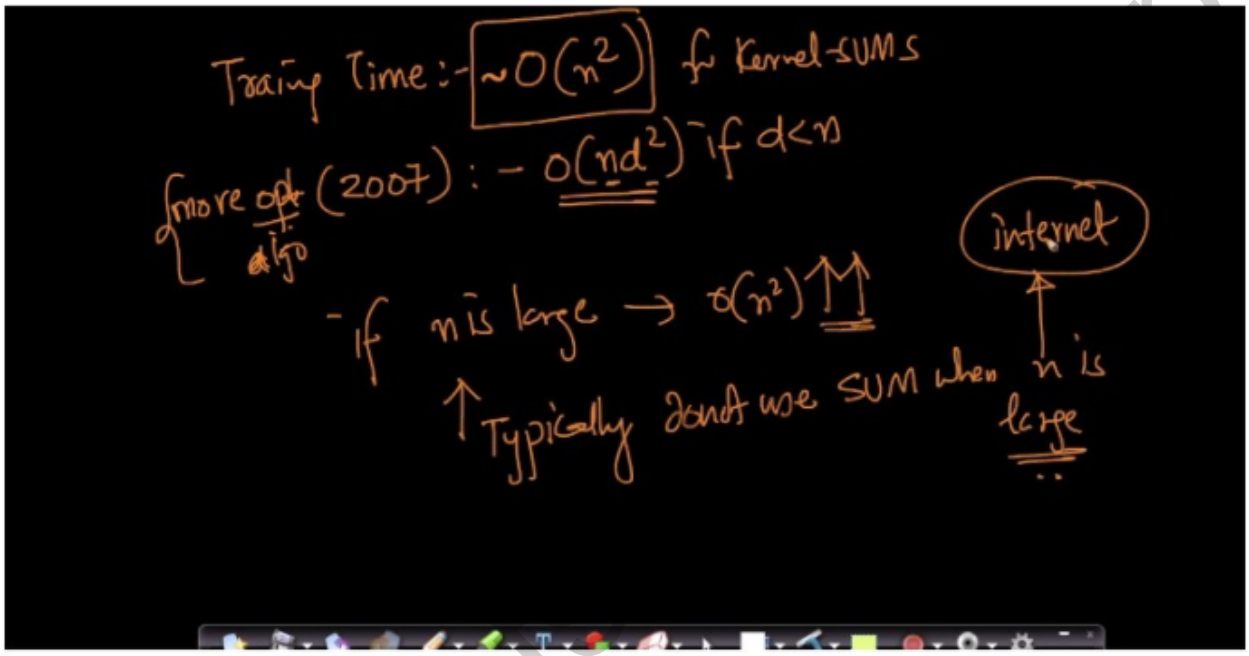


Timestamp 2:10

Here we will look at the time complexity of SVM.

We can train our SVM using gradient descent. However there is a specialized algorithm which solves the dual problem called the SMO or Sequential Minimal Optimization which does a much better job at solving it. We won't go into the details of this.

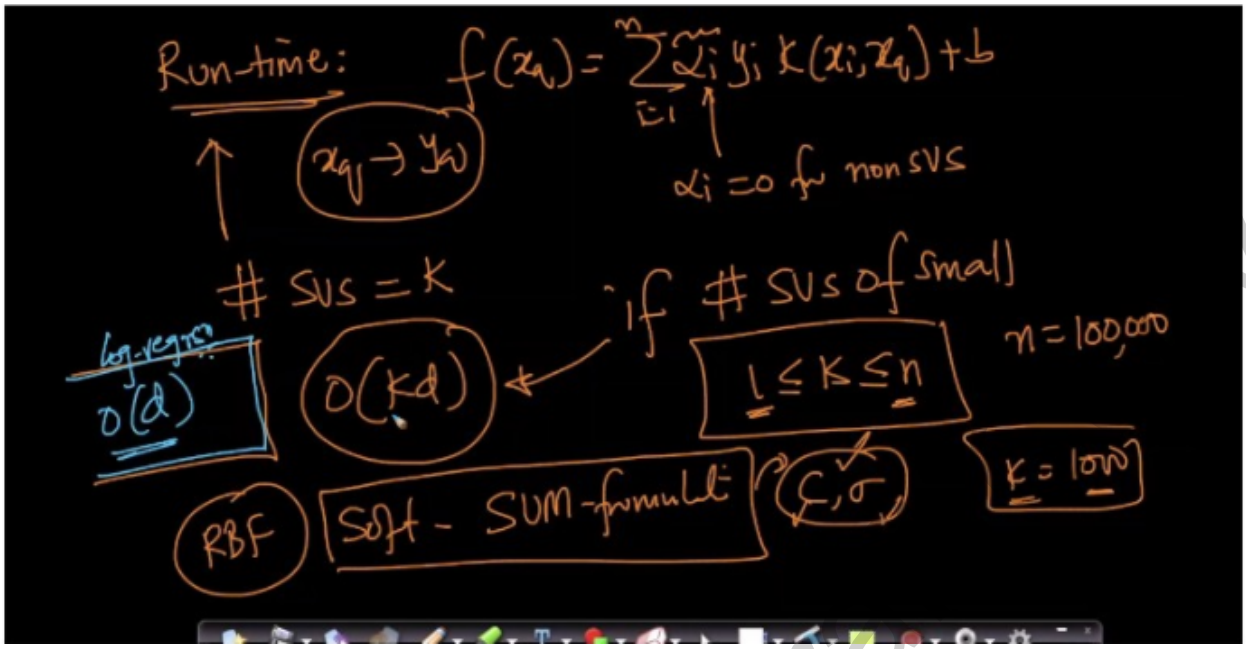
libSVM is a library which has a very good implementation of SVM. It is very optimized. Sklearn also has an implementation of SVM.



Timestamp 4:23

Train time complexity for kernel SVM is around $O(n^2)$, this is not exact.
So, in places where n is large we typically don't use SVM like internet applications. However this is not mandatory.

In 2007 there has been a more optimized algorithm which has train time complexity of $O(nd^2)$, if d < n. d is the number of dimensions.



Timestamp 7:11

The run time complexity for SVM

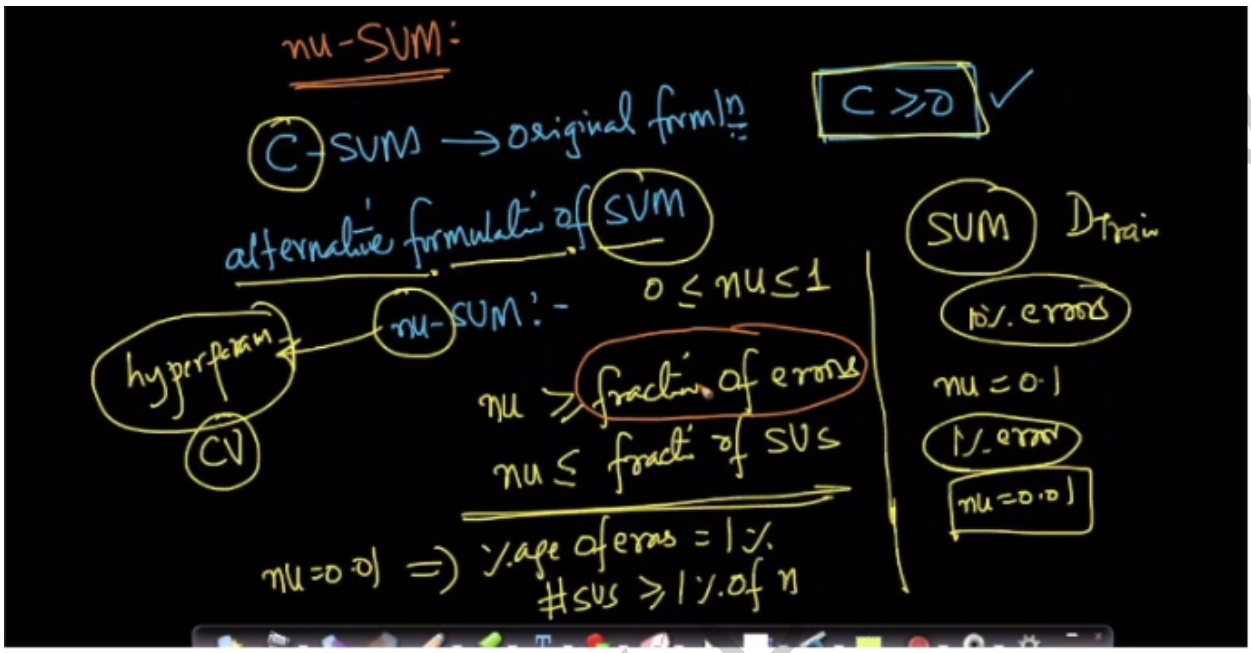
$f(x_q) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x_q) + b$

is O(kd), where k is the number of support vectors and d is the dimension of a datapoint. Remember that α_i's are 0 for non support vectors.

Now, in Soft Margin SVM we don't have control over the number of support vectors we have. So in cases where we have a large number of support vectors the run time complexity becomes higher. Making it not suitable for low latency applications.
However we will look at some techniques where we try to control the number of support vectors but they are not that commonly used.

In logistic regression the run time complexity was O(d), which typically is lesser than the run time complexity of kernel svm's.

36.11 nu-SVM: control errors and support vectors

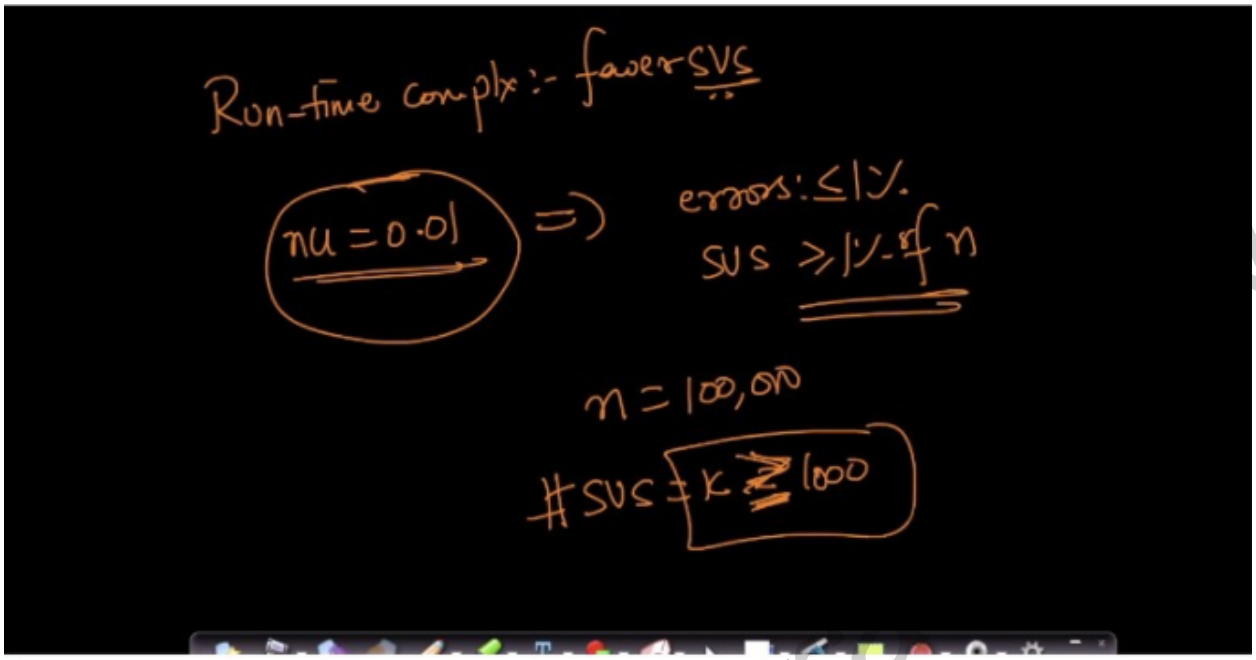


Timestamp 3:10

nu-SVM is an alternative formulation of SVM, where nu is the hyperparameter. The original SVM is also called C-SVM because the hyperparameter C. $C \geq 0$.

Now, this nu-SVM has this hyperparameter nu which lies between 0 and 1. $0 \leq nu \leq 1$. This hyperparameter controls the fractions of errors and the fraction of support vectors.

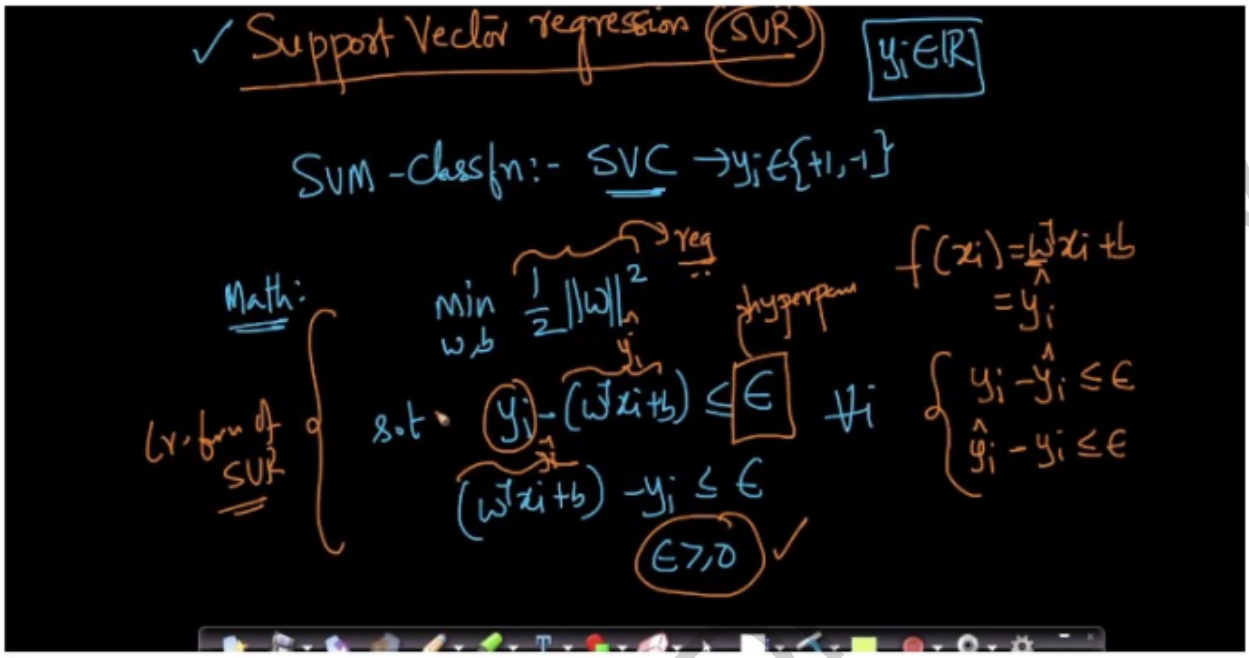
More specifically, $nu \geq$ fraction of errors, and $nu \leq$ fraction of support vectors.
Say we have nu = 0.01 then it implies that fraction of errors $\leq$ 1% of n and number of support vectors $\geq$ 1% of n.



Timestamp 5:30

If we carefully observe the nu-SVM formulation we can see that there is no upper limit to the number of support vectors. There is a limit to errors but not support vectors. So using nu-SVM can give us a fair bit of understanding about the number of support vectors that we can see. But not control over the number of support vectors instead.
Proofing the equivalence between nu-SVM and C-SVM is beyond the scope of this course.

36.12 SVM Regression



Timestamp 2:40

SVM's can be used for both classification and regression. If we use it for classification we generally call it Support Vector Classifier where $y_i \epsilon \{ +1, -1\}$. If we use it for regression we generally call it Support Vector Regressor where $y_i \epsilon \Re$.

The mathematical formulation for SVR is :

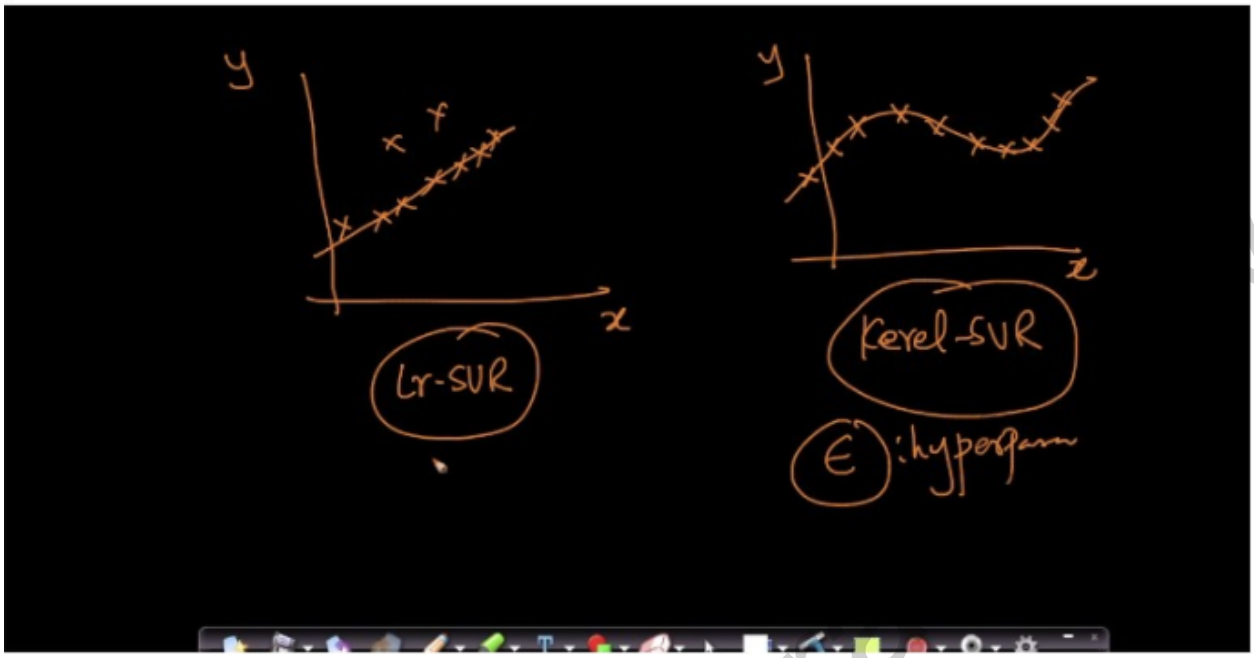
$(w^*, b^*) = argmin_{(w,b)}$ 1/2 $|| w ||^2$,

s.t. ∀i , y_i - $(w^T x_i + b) \leq \varepsilon$

$(w^T x_i + b) - y_i \leq \varepsilon$

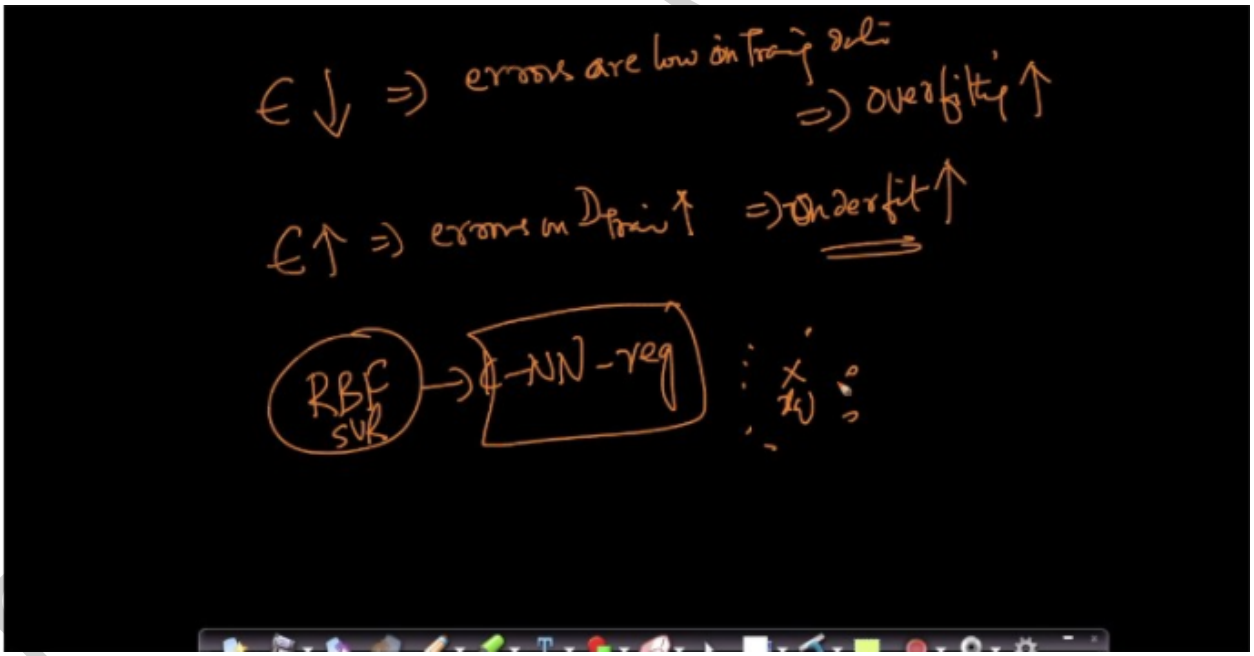
$\varepsilon \geq 0$

The constraints essentially means that the difference between the predicted value and the actual value should less than or equal to ε , as you can see from the image above. Here, ε is a hyperparameter. This is essentially the linear formulation of SVR which can also be kernelized.



Timestamp 3:27

As we can see from the image above. Linear SVR can only fit linear hyperplanes. To fit a non-linear hyperplane we need kernelization.

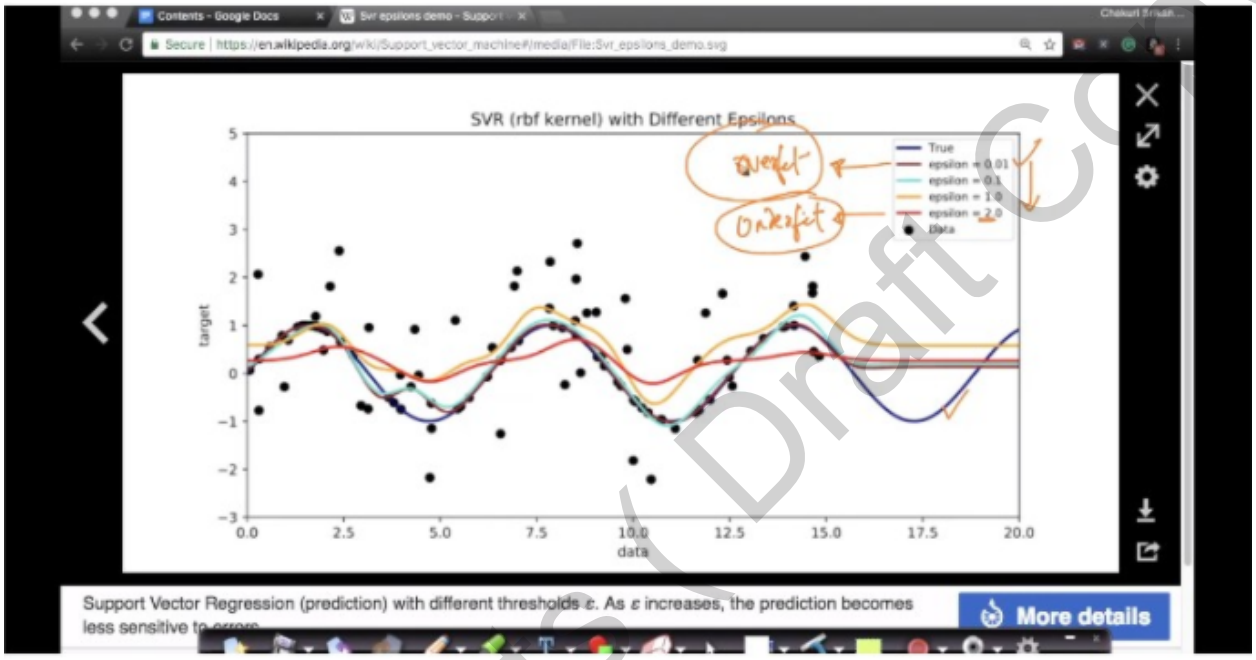


Timestamp 4:30

So we have our hyperparameter ε . If the value of ε is decreased this effectively means that we want lesser errors on training data. Resulting in overfitting.

Similarly if the value ε is increased that means we can tolerate high errors, that means underfitting.

Roughly RBF-SVR behaves similarly to KNN-Regression.



Timestamp 6:12

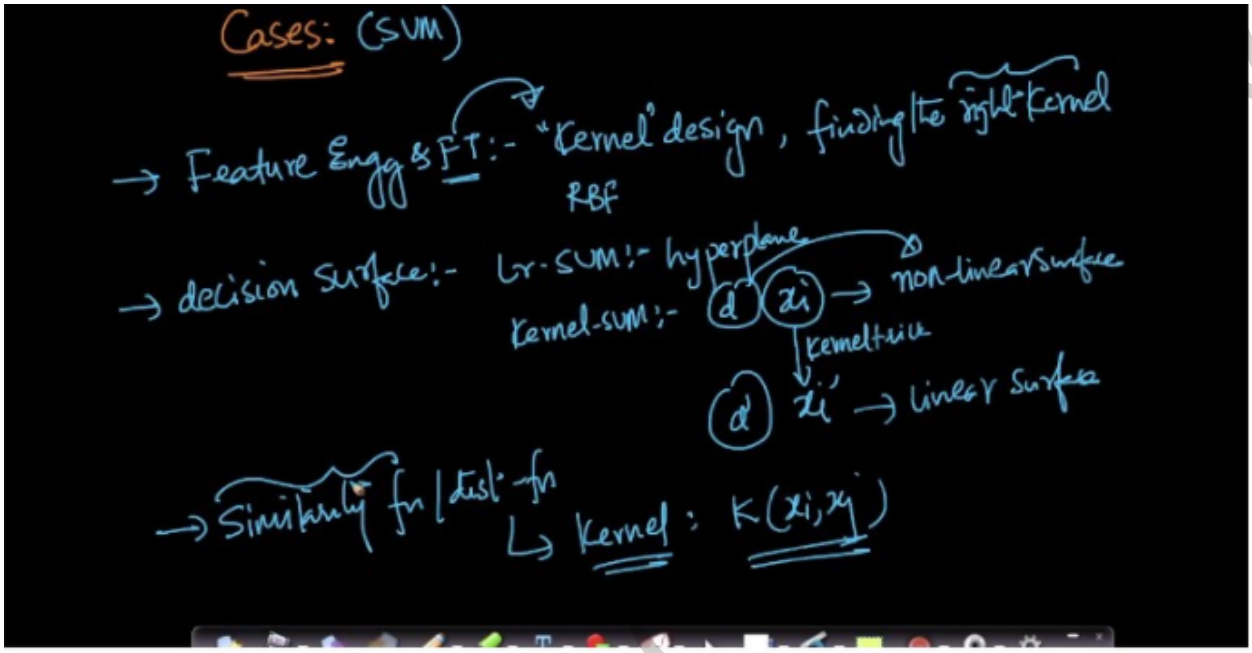
We can see from the image above, how SVR based on rbf kernel behaves for different values of ε . For lower values it is overfitting and for higher values it is underfitting.

You can also see these:

Confusion regarding SVR constraints: https://youtu.be/kgu53eRFERc

For more mathematical details, check out https://alex.smola.org/papers/2004/SmoSch04.pdf

36.13 Cases



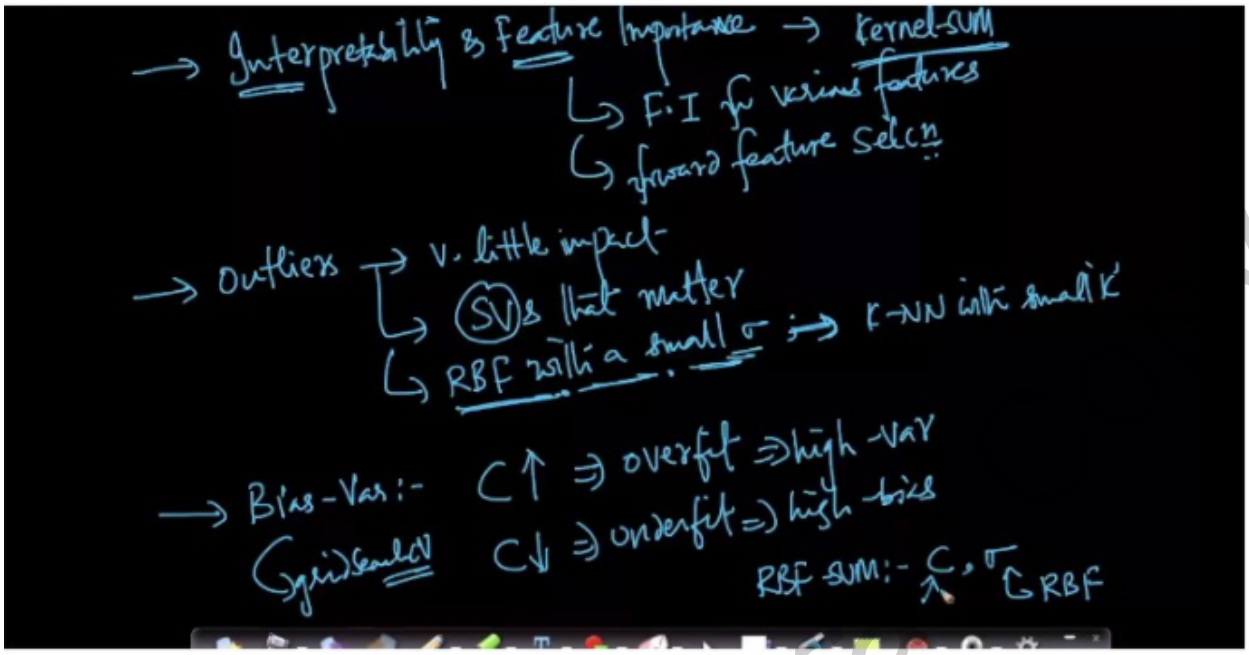
Timestamp 2:40

Here we will look at various cases for SVM:

1. Feature Engineering and Feature Transformation: We have seen that we can do this using kernels. The trick is to find the right kernels. Of course in general we can use the rbf-kernel.

2. Decision Surface: If we are using a linear SVM then the decision surface will always be linear. In order to fit data on a non-linearly separable data we can do kernelization. In kernelization we essentially take our data into higher dimensions in the hope that in higher dimensions we can find some hyperplane which can separate our data.

3. Similarity / Distance function: We can very easily use any similarity or distance function in SVM. Kernels are essentially similarity/distance measures.

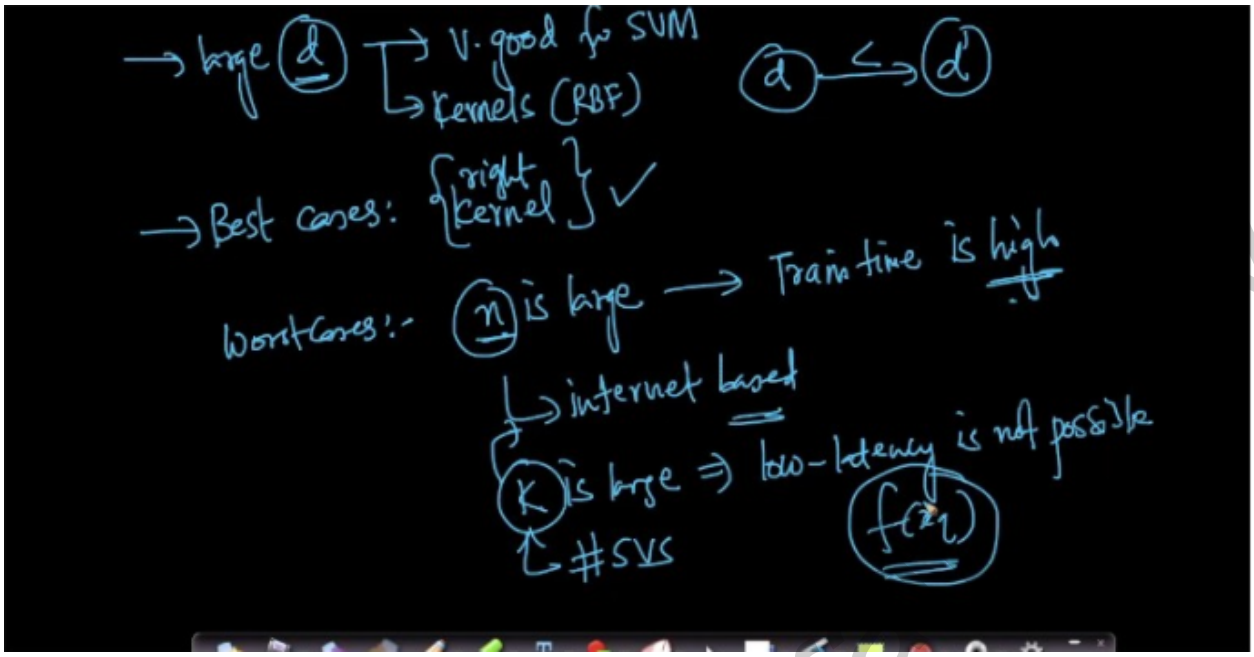


Timestamp 6:00

4. Interpretability and Feature Importance: SVM's are not that interpretable especially Kernel SVM's. Also it is difficult to find feature importance directly. We can use techniques like Forward/Backward feature selection but they are time consuming.

5. Outliers: SVM's are not much impacted by outliers, because only support vectors matter. However if we use a rbf kernel with a very low σ, then there can be overfitting. We can think of this as choosing a small k in KNN.

6. Bias - Variance: In SVM's C is our hyperparameter. If the value of C increased we have overfitting i.e. high variance. If it's value is decreased then we have underfitting or high bias.



Timestamp 8:27

7. Large d: Large dimensions are very good for SVM as with kernelization we are essentially taking our data points to higher dimensions.

So, we will have the best cases when we have the right kernel.
Worst cases will occur when we have large n, this increases our training time. Large n's are typically seen in internet applications. Large k i.e. no of support vectors is also an issue, because it increases our test time complexity considerably. Cases where low-latency is the requirement this can be an issue.

Typically when we have a large n we use logistic regression with feature transformation/engineering.

36.14 Code Sample

We can see the scikit learn's implementation of SVM and its variation here =>

https://scikit-learn.org/stable/modules/svm.html