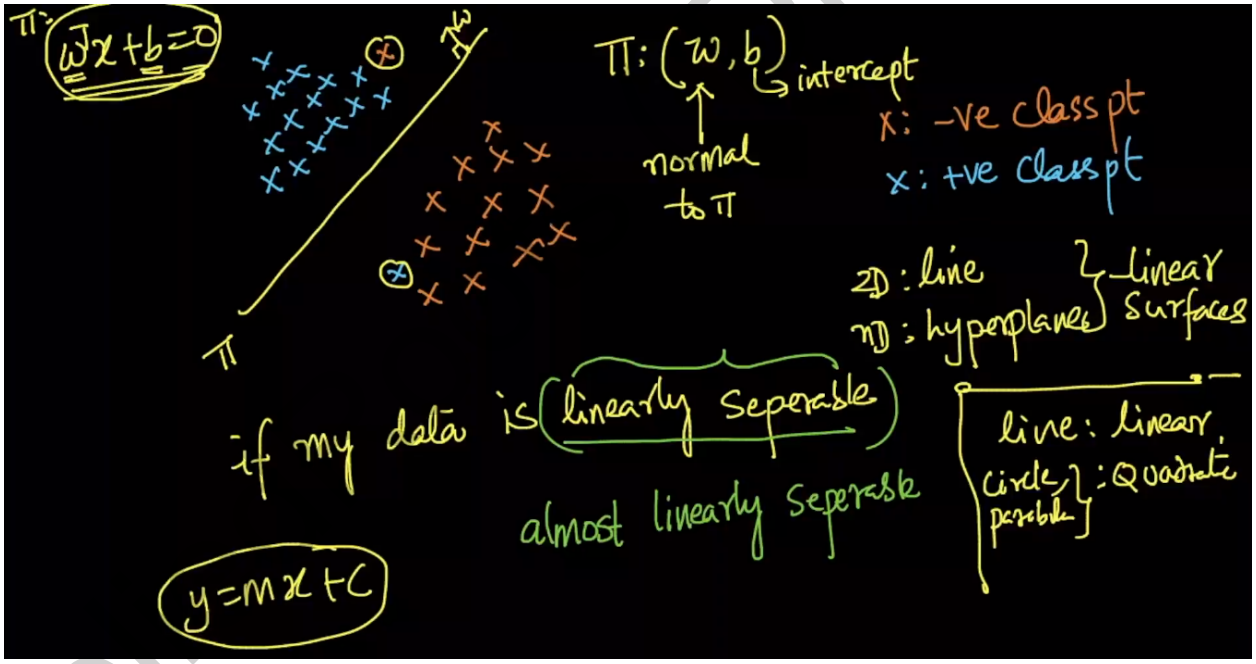
33.1 - Geometric Intuition of Logistic Regression

Logistic Regression is one of the classification techniques used in Machine Learning. So far we have seen Naive Bayes, which was a Probability oriented technique, whereas Logistic Regression is a geometry-oriented technique.

Logistic Regression can be derived using any one of the 3 below perspectives. They are

a) Geometry
b) Probability
c) Loss Function

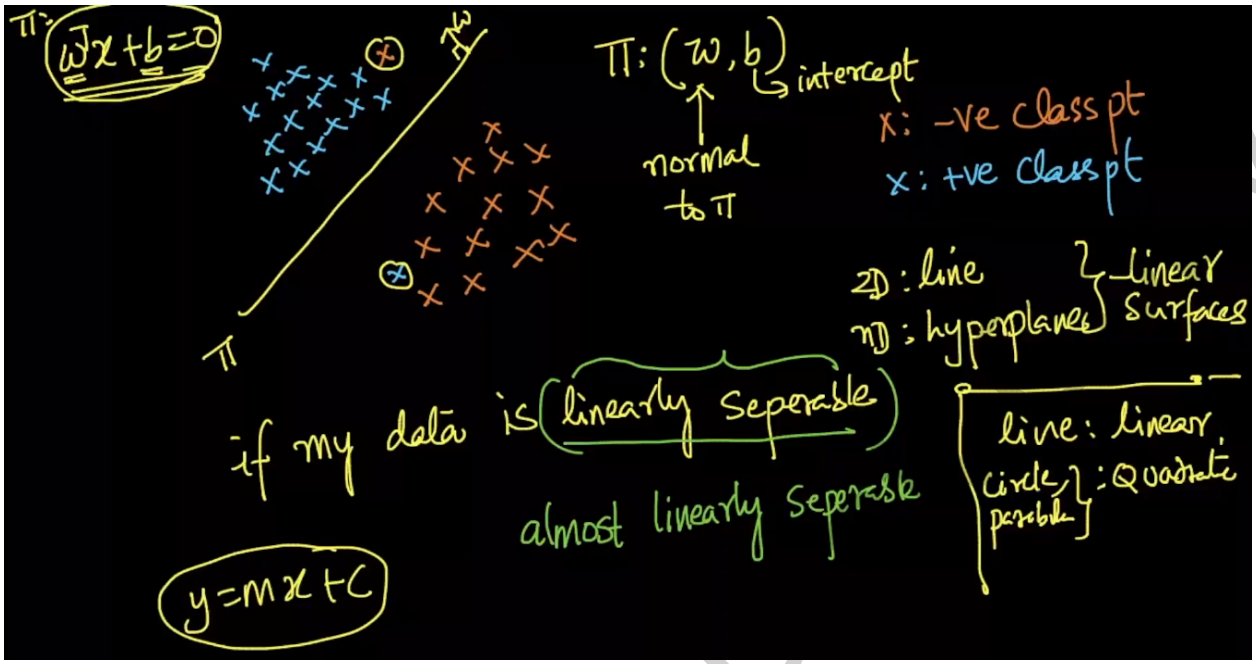
Let us assume we have the points belonging to 2 classes(positive and negative), as shown in the figure below explained starting from the timestamp 3:20.



If the data points of different classes can be separated by using a linear surface, then we say the data is linearly separable. The linear surfaces in different dimensions are called as mentioned below

2D → Line
3D → Plane
nD → Hyperplane

Circle, Parabola, Ellipse, Sphere are the examples of Quadratic Surfaces.



In the above representation of the points,

π → Hyperplane

w → Perpendicular to the plane

The equation of the hyperplane 'π' is represented in the form of 'w' as $\mathbf{w^Tx+b = 0}$ where b → intercept.

Here $\mathbf{x} \in \mathbf{R^d}$, $\mathbf{w} \in \mathbf{R^d}$, $\mathbf{b} \in \mathbf{R^1}$.

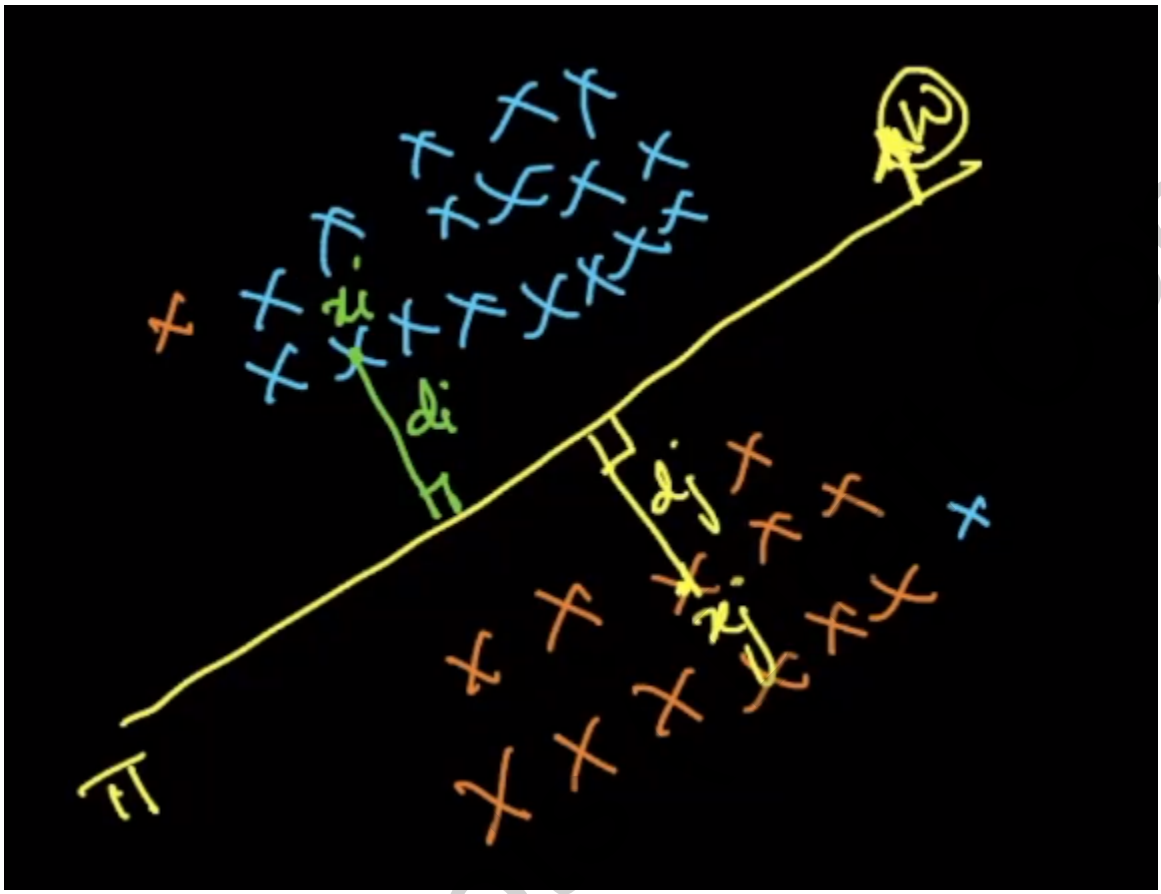
If the hyperplane 'π' passes through the origin, then b=0 and the equation becomes $\mathbf{w^Tx = 0}$.

Assumption of Naive Bayes: All the features are conditionally independent of each other.

Assumption of K-NN: Neighborhood is the same as the query point.

Assumption of Logistic Regression: All the classes in the dataset are almost/perfectly linearly separable.

Geometric Intuition



For all the positive points, let $y_i = +1$
For all the negative points, let $y_i = -1$

So far, we have seen
If $y_i = 1 \rightarrow$ class = +ve
If $y_i = 0 \rightarrow$ class = -ve

But for now,
If $y_i = 1 \rightarrow$ class = +ve
If $y_i = -1 \rightarrow$ class = -ve

So for Logistic Regression, $\mathbf{y_i \in \{-1, +1\}}$
Distance of the point 'x_i' to the hyperplane 'π' = $\mathbf{d_i = w^T x_i / ||w||}$
Let us assume 'w' is a unit vector, so then $\mathbf{||w|| = 1}$.

So now the distance '$\mathbf{d_i}$' becomes $\mathbf{w^T x_i}$
From the above figure, as 'w' and 'x_i' are on the same side of the hyperplane 'π', we have $\mathbf{w^T x_i > 0}$
As 'w' and 'x_j' are on the opposite sides of the hyperplane 'π', we have $\mathbf{w^T x_j < 0}$

So for any point 'x_q', the classifier works as
If $\mathbf{w^T x_q > 0}$, then $\mathbf{y_q' = +1}$
If $\mathbf{w^T x_q < 0}$, then $\mathbf{y_q' = -1}$

Different Cases in classification using Logistic Regression

Case 1: If $\mathbf{y_i = +1}$ and $\mathbf{w^T x_i > 0}$

It means the point 'x_i' is actually a positive point and is also classified as positive. Hence the classifier has classified this data point correctly.

Case 2: If $\mathbf{y_i = -1}$ and $\mathbf{w^T x_i < 0}$

It means the point 'x_i' is actually a negative point and is also classified as negative. Hence the classifier has classified this data point correctly.

Case 3: If $\mathbf{y_i = +1}$ and $\mathbf{w^T x_i < 0}$

It means the point 'x_i' is actually a positive point but is classified as negative. This point has been misclassified.

Case 4: If $\mathbf{y_i = -1}$ and $\mathbf{w^T x_i > 0}$

It means the point 'x_i' is actually a negative point but is classified as positive. This point has been misclassified.

Note
If a point (x_i, y_i) is classified correctly, then $\mathbf{y_i * (w^T x_i) > 0}$.
If a point (x_i, y_i) is misclassified, then $\mathbf{y_i * (w^T x_i) < 0}$.
For a classifier to be very good, the number of misclassified points should be minimum, and the number of correctly classified points should be maximum.

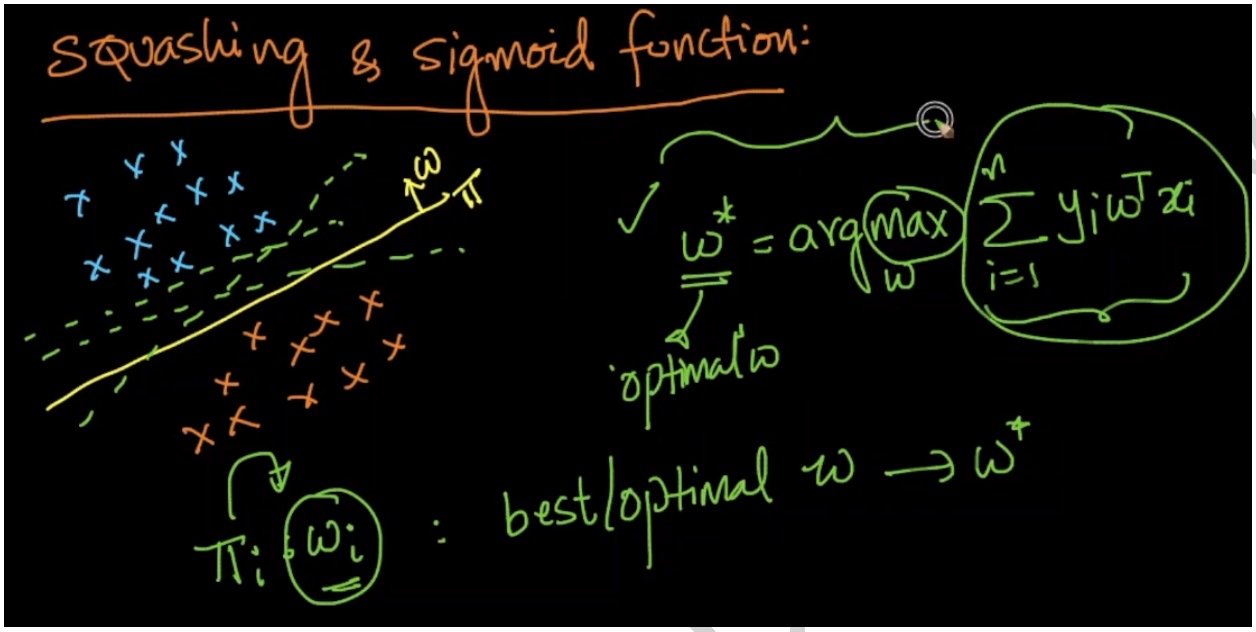
So the objective is to have as many points as possible with $\mathbf{y_i*(w^Tx_i)>0}$.
So we need to maximize $\mathbf{\Sigma_{i=1}^{n}\ y_i*(w^Tx_i)}$
Where 'n' → total number of points in the training dataset (D_{Train}).

But from the given dataset, 'x_i' and 'y_i' are fixed. We can only find 'w' that is variable. So we can find an optimal 'w' that could maximize $\mathbf{\Sigma_{i=1}^{n}\ y_i*(w^Tx_i)}$.

Let **w*** be the optimal 'w' that could maximize $\mathbf{\Sigma_{i=1}^{n}\ y_i*(w^Tx_i)}$.
$\mathbf{w^* = arg\text{-}max(\Sigma_{i=1}^{n}\ y_i*(w^Tx_i))}$
This is the mathematical optimization problem we have to solve, in order to obtain the optimal 'w'.

$\mathbf{y_i*(w^Tx_i) = +ve}$ → If 'x_i' is correctly classified.
$\mathbf{y_i*(w^Tx_i) = -ve}$ → If 'x_i' is mis-classified.

33.2 - Sigmoid Function: Squashing



The objective of Logistic Regression is to find a plane 'π' with an optimal 'w' that could maximize $\mathbf{\Sigma_{i=1}^{n}\ y_i*(w^Tx_i)}$

The function that has to be maximized is **arg-max** $\mathbf{\Sigma_{i=1}^{n}\ y_i*(w^Tx_i)}$

$\mathbf{w^Tx_i}$ → distance from the point 'x_i' to the plane 'π'.

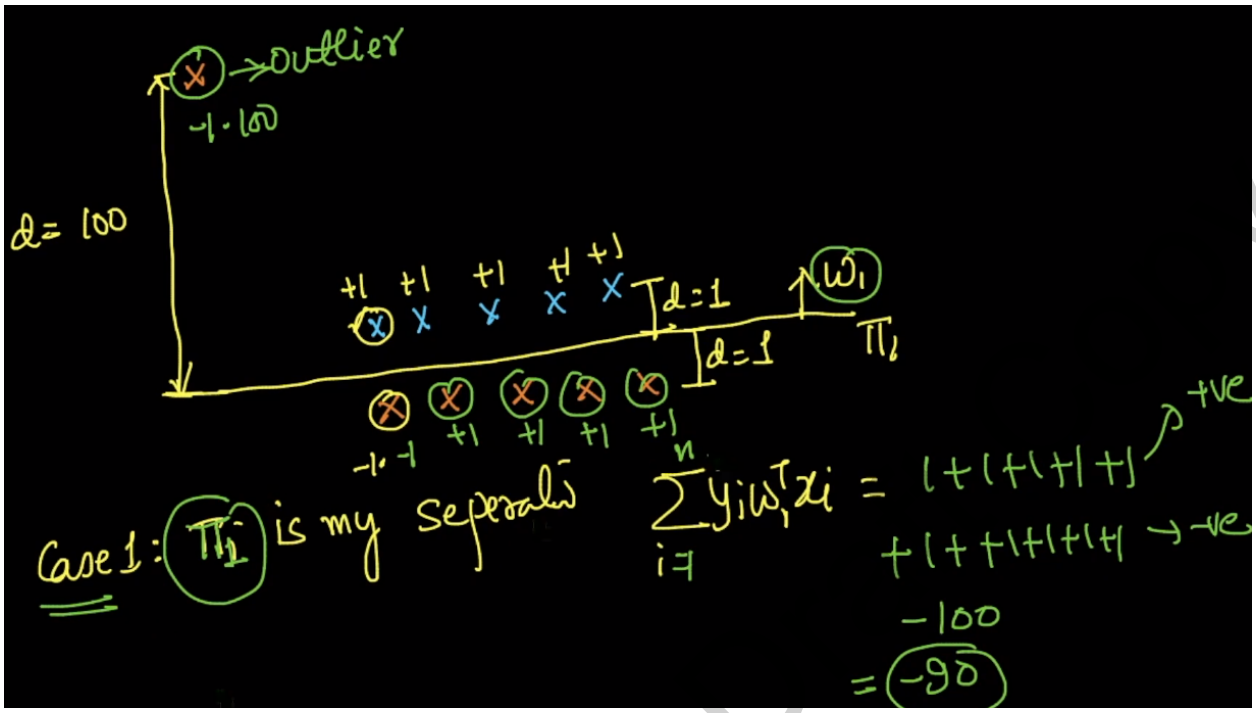
$\mathbf{y_i \in \{-1, +1\}}$

If $\mathbf{y_i*(w^Tx_i)}$ is +ve → 'π' as defined by 'w' correctly classifies 'x_i'.

If $\mathbf{y_i*(w^Tx_i)}$ is -ve → 'π' as defined by 'w' incorrectly classifies 'x_i'.

We have to maximize the number of correctly classified points and minimize the incorrectly classified points.

Case 1

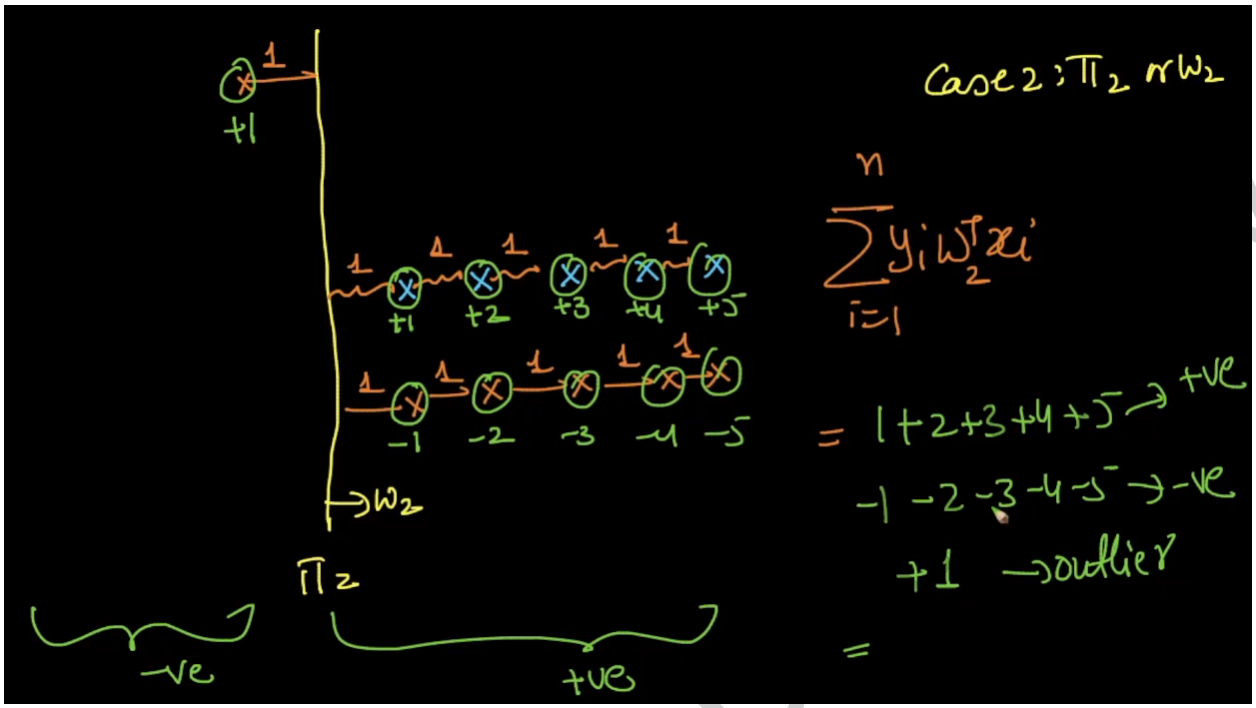


In the above figure, we can see the 5 positive points are present in the same side as that of the vector 'w'. We also can see the 5 negative points present on the opposite side of the vector 'w'.
So for all the given 5 '+ve' points, $w^T x_i = 1$ and $y_i = 1$.
So for all the given 5 '+ve' points, $y_i*(w^T x_i) = 1 * 1 = 1$

For all the given 5 '-ve' points, $w^T x_i = -1$ and $y_i = -1$
So for all the given 5 '-ve' points, $y_i*(w^T x_i) = (-1) * (-1) = 1$

We have a '-ve' point present on the same side as vector 'w' and is located at a distance of 100 units from the plane 'π'.
So $y_i*(w^T x_i) = -1 * 100 = -100$
So now,
$\Sigma_{i=1}^{n} y_i*(w^T x_i) = (1+1+1+1+1) + (1+1+1+1+1) + (-100) = -90$

Case 2



So here,

$\Sigma_{i=1}^{n} y_i*(w^T x_i) = (1+2+3+4+5)+(-1-2-3-4-5)+1 = 1$

As we know the objective o Logistic Regression is to maximize the value of $\mathbf{\Sigma_{i=1}^{n} y_i*(w^T x_i)}$, with the planes '$\pi_1$' and '$\pi_2$', we get the sum values as -90 and 1 respectively. So if we want to go with the maximum value of $\Sigma_{i=1}^{n} y_i*(w^T x_i)$, we have to go for case 2.

But from the geometric intuition point, in case 1, out of 11 points, 10 of them are classified correctly, whereas in case 2, only 6 of them are classified correctly.

Accuracy in case 1 = 10/11 = 0.909

Accuracy in case 2 = 6/11 = 0.545

Intuitively we can say 'π_1' is better than 'π_2'. We are getting a larger value of $\Sigma_{i=1}^{n} y_i*(w^T x_i)$ in case 2, just because of the outlier point, which is very dar away, and it makes us prefer the plane 'π_2'. Here one single outlier is impacting our model drastically.

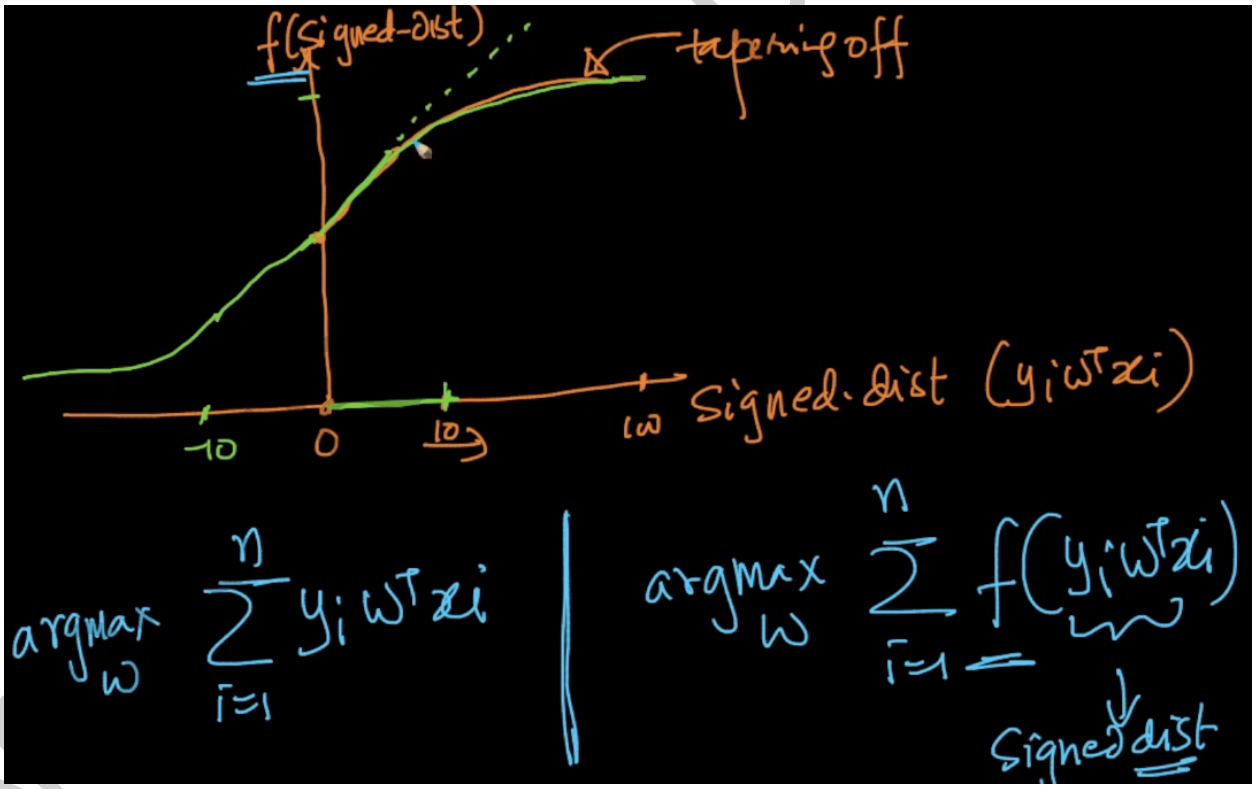
So the maximization of summation of signed distances is much prone to the outliers. It means the outliers can impact a lot.

So we have to modify this formulation, by modifying the function **arg-max$_w$ $\Sigma_{i=1}^{n}$ y_i*($w^T x_i$)** in such a way that the outliers do not impact our model. For this purpose, we use a technique called **Squashing**.

Squashing

The idea of squashing is instead of using the signed distances as they are for all the points,
If signed distance is small → Use the signed distance as it is.
If signed distance is large → Make it a smaller value
So in our example,
If the signed distance of the normal point to the plane 'π_1' is 1 unit, then we can use it as it is. Whereas if the signed distance of an outlier point to the plane 'π_1' is 100 units, we have to reduce it to a smaller value.

How to reduce the large value to a smaller one?



Let us assume we have all the values of signed distances on X-axis(ie., the values of **y_i*($w^T x_i$)**). Let the 'Y' axis be a function of the signed distances (ie., function of **y_i*($w^T x_i$)**).

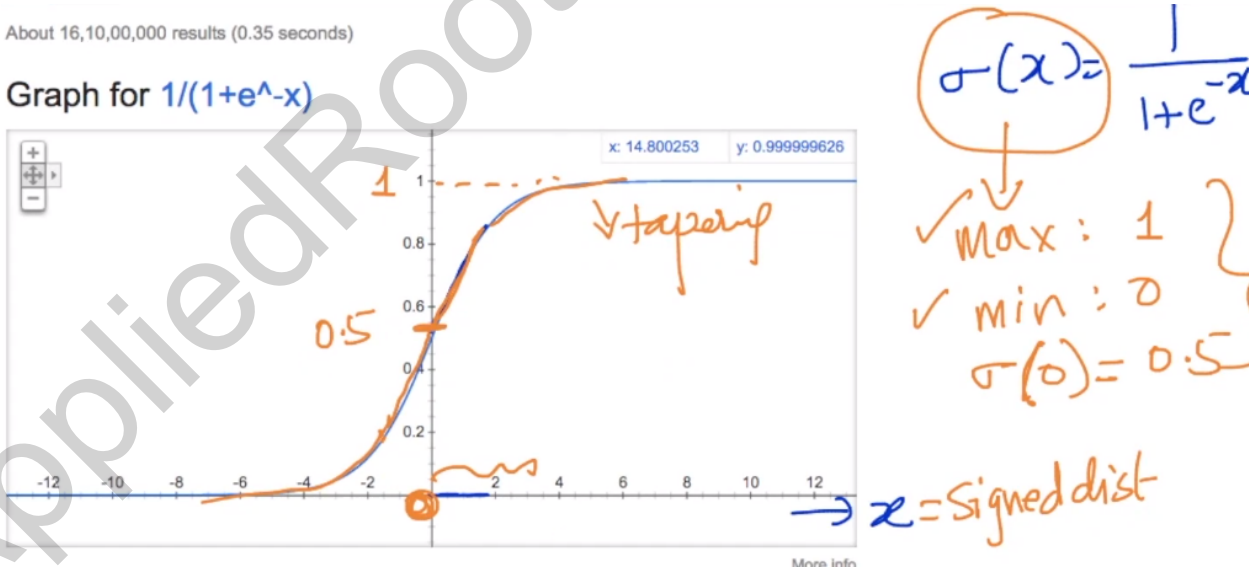
Now we should make sure that from $\mathbf{y_i*(w^Tx_i) = 0}$, as the value of $\mathbf{y_i*(w^Tx_i)}$ increases, we want $\mathbf{f(y_i*(w^Tx_i))}$ to increase linearly.

This function $\mathbf{f(y_i*(w^Tx_i))}$ needs to increase linearly for smaller positive values of $\mathbf{y_i*(w^Tx_i)}$ and should taper-off for larger positive values of $\mathbf{y_i*(w^Tx_i)}$. Here tapering-off means becoming constant.

We should make sure this function increases linearly upto a certain threshold value of $\mathbf{y_i*(w^Tx_i)}$ on the positive side of the 'X' axis and then gets tapered off. Similarly on the negative side of the 'X' axis, the function should decrease linearly upto a threshold value and then should get tapered off.

So here in order to get rid of the problem of outliers, we are converting the problem of $\mathbf{arg\text{-}max_w \ \Sigma_{i=1}^n \ y_i*(w^Tx_i)}$ to $\mathbf{arg\text{-}max_w \ \Sigma_{i=1}^n \ f(y_i*(w^Tx_i))}$.

Here we actually need a function $\mathbf{f(y_i*(w^Tx_i))}$ in such a way that on the positive side of the axis of $\mathbf{y_i*(w^Tx_i)}$, it increases linearly upto a certain threshold value and then gets tapered off. Similarly on the negative side of the axis of $\mathbf{y_i*(w^Tx_i)}$, it decreases linearly upto a certain threshold value and then gets tapered off. Such a function satisfying our requirement is **Sigmoid Function**.



Sigmoid function: $\mathbf{\sigma(x) = 1/(1+\exp(-x))}$

So now we shall change the mathematical formulation to $\mathbf{arg\text{-}max_w \ \Sigma_{i=1}^n \ \sigma(y_i*(w^Tx_i))}$

Maximum value of $\sigma(x) = 1$
Minimum value of $\sigma(x) = 0$
$\sigma(0) = 0.5$

So in our problem, we have seen outlier present at a distance of 100 units. Here as the maximum value of the sigmoid function is 1, the distance of outlier instead of 100 units, gets tapered off to 1 unit.

If a point (x_i, y_i) lies on the hyperplane 'π', then $\mathbf{w^T x_i = 0}$.
$P(y_i*(w^T x_i)) = P(0) = 0.5$
So $P(y_i=1) = 0.5$

If a point (x_i, y_i) lies on the same side as 'w' of the hyperplane 'π', then $\mathbf{w^T x_i > 0}$.
$\sigma(y_i*(w^T x_i)) > 0.5$
So $P(y_i=1) > 0.5$

If a point (x_i, y_i) lies on the opposite side as 'w' of the hyperplane 'π', then $\mathbf{w^T x_i > 0}$.
$\sigma(y_i*(w^T x_i)) < 0.5$
So $P(y_i=1) < 0.5$

Note: In all the above 3 cases, (x_i, y_i) is a '+ve' point.

So we have a good probabilistic interpretation with Sigmoid function. Coming back to our problem, our actual mathematical formulation was to maximize the sum of signed distances (ie., $\mathbf{arg\text{-}max_w \Sigma_{i=1}^n y_i*(w^T x_i)}$), but it was severely impacted by the outliers.

Hence we have gone for sigmoid function which has got the below properties required for our problem.

1) Linear Behaviour for small values of $\mathbf{y_i*(w^T x_i)}$.
2) Tapering Behavious for large values of $\mathbf{y_i*(w^T x_i)}$.
3) Nice Probabilistic Interpretation

So now our mathematical formulation is to maximize the sum of transformed signed distances.

So the optimal value of 'w' is the value that maximizes

$$\mathbf{w^* = arg\text{-}max_w \left(\Sigma_{i=1}^{n} \sigma(y_i^*(w^T x_i))\right) = arg\text{-}max_w \left(\Sigma_{i=1}^{n} 1/(1+\exp(-y_i^*(w^T x_i)))\right)}$$

This above mathematical formulation is less impacted by the presence of outliers, when compared to the sum of signed distances. This function is also called Squashing and it helps us in getting rid of outliers. Here squashing reduces the $\mathbf{f(y_i^*(w^T x_i))}$ values which lie in $(-\infty, \infty)$ to the values in [0,1].

33.3 - Mathematical Formulation of Objective Function

The optimization problem is $w^* = \arg\text{-}\max_w \Sigma_{i=1}^{n} 1/(1+\exp(-y_i*(w^T x_i)))$

Here we shall use monotonic functions in this optimization problem.

A function g(x) is said to be monotonic

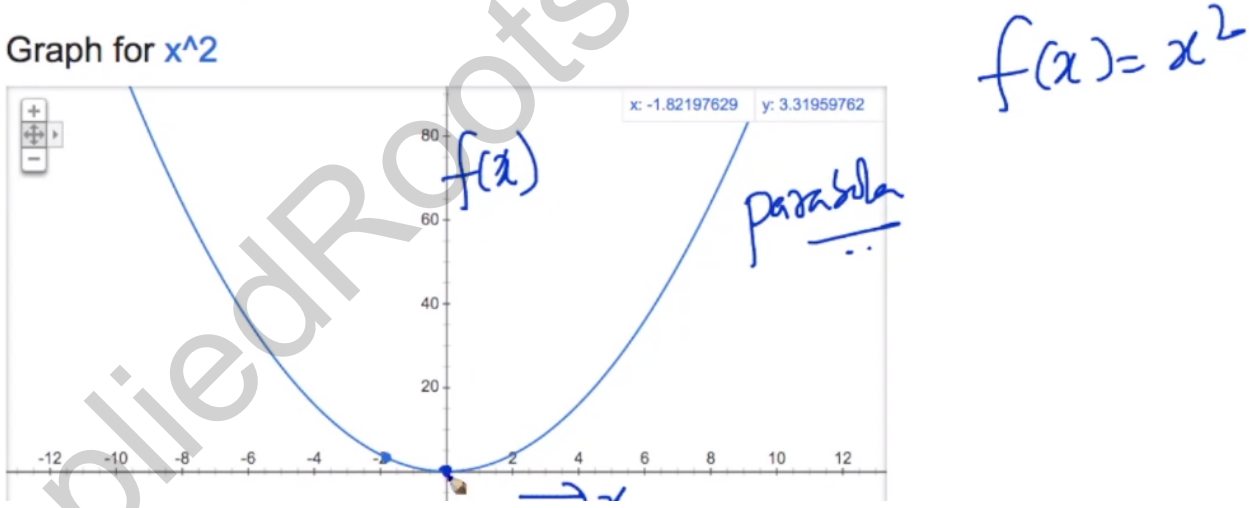
a) If g(x) increases with an increase in 'x', then g(x) is called monotonically increasing function.
b) If g(x) decreases with an increase in 'x', then g(x) is called monotonically decreasing function.

For all $x_1>x_2$, if $g(x_1)>g(x_2)$, then we can say g(x) is monotonically increasing function.

For all $x_1>x_2$, if $g(x_1)<g(x_2)$, then we can say g(x) is monotonically decreasing function.

For example, log(x) is defined only for x>0 and is an example of monotonically increasing function. Let us assume we have an optimization problem as below

$x^* = \arg\min_x (x^2)$



The curve of x^2 is a parabola $y=mx^2$ passing through origin. So here we see the minimum value of this curve occurs at x=0. So the optimal value of 'x' in order to minimize this optimization problem is x=0.

The curve $y=x^2$ is

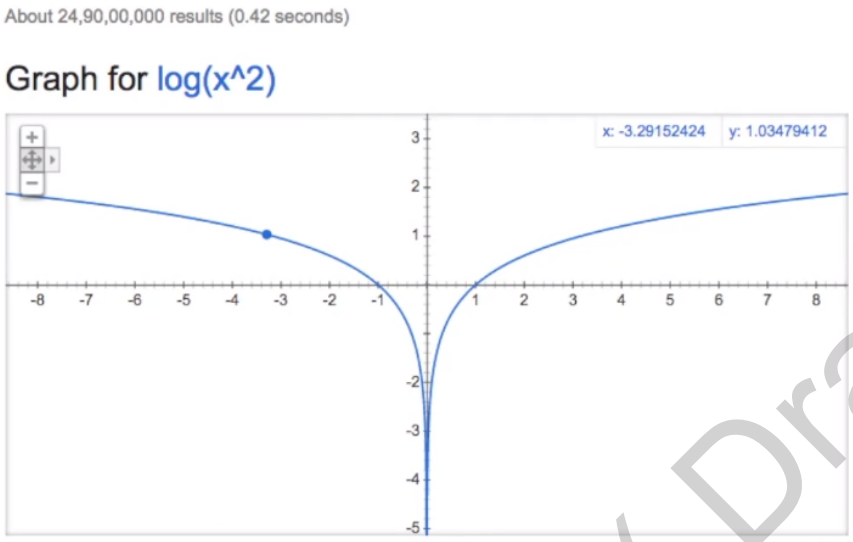
a) Monotonically increasing for x>0
b) Monotonically decreasing for x<0

So we have assumed g(x) = log(x) which is increasing function. So far we have worked on the optimization problem of $\mathbf{x^* = arg\text{-}min_x\ f(x)}$

$f(x) = x^2$

Now we shall look at the optimization problem of $\mathbf{x^l = arg\text{-}min_x\ g(f(x))}$

As g(x) = log(x) → g(f(x)) = $g(x^2) = \log(x^2)$. The graph of $\log(x^2)$ is given as



The minimum value of $\log(x^2)$ also occurs at x=0.

Let us assume we have a function f(x) and a monotonic function g(x), then

$\mathbf{arg\text{-}min_x\ f(x) = arg\text{-}min_x\ g(f(x))}$

$\mathbf{arg\text{-}max_x\ f(x) = arg\text{-}max_x\ g(f(x))}$

These above two properties are valid only if g(x) is a monotonic function. Coming back to our original problem,

$\mathbf{w^* = arg\text{-}max_w\ \Sigma_{i=1}^{n}\ 1/(1+exp(-y_i*(w^Tx_i)))}$

Let us consider g(x) = log(x) which is a monotonic function. So from the property of monotonic function, we can write the optimization problem as

$\mathbf{w^* = arg\text{-}max_w\ \Sigma_{i=1}^{n}\ log(1/(1+exp(-y_i*(w^Tx_i))))}$

We have a formula that **log(1/x) = -log(x)**

So the optimization problem now becomes

$\mathbf{w^* = arg\text{-}max_w\ \Sigma_{i=1}^{n}\ -log(1+exp(-y_i*(w^Tx_i)))}$

Now arg-max (f(x)) = arg-min (-f(x))

So the optimization $\mathbf{w^* = arg\text{-}max_w \Sigma_{i=1}^n -log(1+exp(-y_i*(w^Tx_i)))}$ changes to $\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n log(1+exp(-y_i*(w^Tx_i)))}$

Finally this is the optimization problem of Logistic Regression with the class labels +1 and -1.

If we do not have the term '1' in the optimization problem, then the optimization problem becomes

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n log(exp(-y_i*(w^Tx_i)))}$

Here the 'log' and 'exp' get cancelled out because $\log_e e^x = x$.

So now the optimization problem becomes

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n (-y_i*(w^Tx_i))}$

So if we remove the '-ve' sign, then arg-min becomes arg-max, and the optimization problem becomes

$\mathbf{w^* = arg\text{-}max_w \Sigma_{i=1}^n (y_i*(w^Tx_i))}$

So we can say that the optimization of Logistic Regression is a slight change made to the optimization of summation of signed distances.

But the optimization of summation of the signed distances is more impacted by the outliers. Hence we go with the optimization of the sigmoid function, as it is not much affected by the outliers.

33.4 - Weight Vector

So far we have seen the optimization problem of

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^Tx_i)))}$

The optimal value w* is called the weight vector. This weight vector is d-dimensional same as our data point $x_i \in R^d$.

$w^* = <w_1, w_2, w_3, \ldots, w_d>$

Here as we have 'd' features, we have the weight component associated with each feature in w*. That's the reason we call it the weight vector.

In Logistic Regression, the decision about the class label 'y_q' for a data point 'x_q' is made on the below criteria.

If $w^Tx_q>0$, then $y_q = 1$

If $w^Tx_q<0$, then $y_q = -1$

When it comes to the probabilistic interpretation, if we want to know $P(y_q=1)$, we have to compute the value of $\mathbf{\sigma(w^Tx_q)}$.

Interpretation of w*

Let 'w_i' → i^{th} component of the weight vector (ie., the weight component associated with the i^{th} feature in the dataset)

'x_{qi}' → i^{th} feature value of the data point 'x_q'.

1) If w_i = '+ve' and and if 'x_{qi}' increases, then the value of the product $\mathbf{(w_i.x_{qi})}$ also increases. It means the value of $(\mathbf{\Sigma_{i=1}^{d}\ w_i*x_{qi}})$ also increases. There by the $\mathbf{\sigma(w^Tx_q)}$ increases, and finally $\mathbf{P(y_q=1)}$ increases.
2) If w_i = '-ve' and and if 'x_{qi}' increases, then the value of the product $\mathbf{(w_i.x_{qi})}$ decreases. It means the value of $(\mathbf{\Sigma_{i=1}^{d}\ w_i*x_{qi}})$ decreases. There by the $\mathbf{\sigma(w^Tx_q)}$ decreases, and finally $\mathbf{P(y_q=1)}$ also decreases. The value of $\mathbf{P(y_q= -1)}$ increases.

3) If w_i = '+ve' and and if 'x_{qi}' decreases, then the value of the product $\mathbf{(w_i.x_{qi})}$ decreases. It means the value of $(\mathbf{\Sigma_{i=1}^{d}\ w_i*x_{qi}})$ decreases. There by the $\mathbf{\sigma(w^Tx_q)}$ decreases, and finally $\mathbf{P(y_q=1)}$ also decreases. The value of $\mathbf{P(y_q= -1)}$ increases.
4) If w_i = '-ve' and and if 'x_{qi}' decreases, then the value of the product $\mathbf{(w_i.x_{qi})}$ increases. It means the value of $(\mathbf{\Sigma_{i=1}^{d}\ w_i*x_{qi}})$ increases. There by the $\mathbf{\sigma(w^Tx_q)}$ increases, and finally $\mathbf{P(y_q=1)}$ also increases. The value of $\mathbf{P(y_q= -1)}$ decreases.

Note

In Logistic Regression formulation of the optimization problem, we have not strictly imposed the constraint that 'w' has to be a unit vector. The mathematical formulation will work even if 'w' is a non unit vector perpendicular to the hyperplane.

The only difference would be that the distance from any point 'x_i' to the hyperplane would be $\mathbf{w.x_i/||w||}$ instead of just $\mathbf{w.x_i}$. So the resulting weight vector after solving the optimization problem while being perpendicular to the hyperplane need not be a unit vector always.

33.5 - L2 Regularization: Overfitting and Underfitting

The optimization problem we have been working on so far is

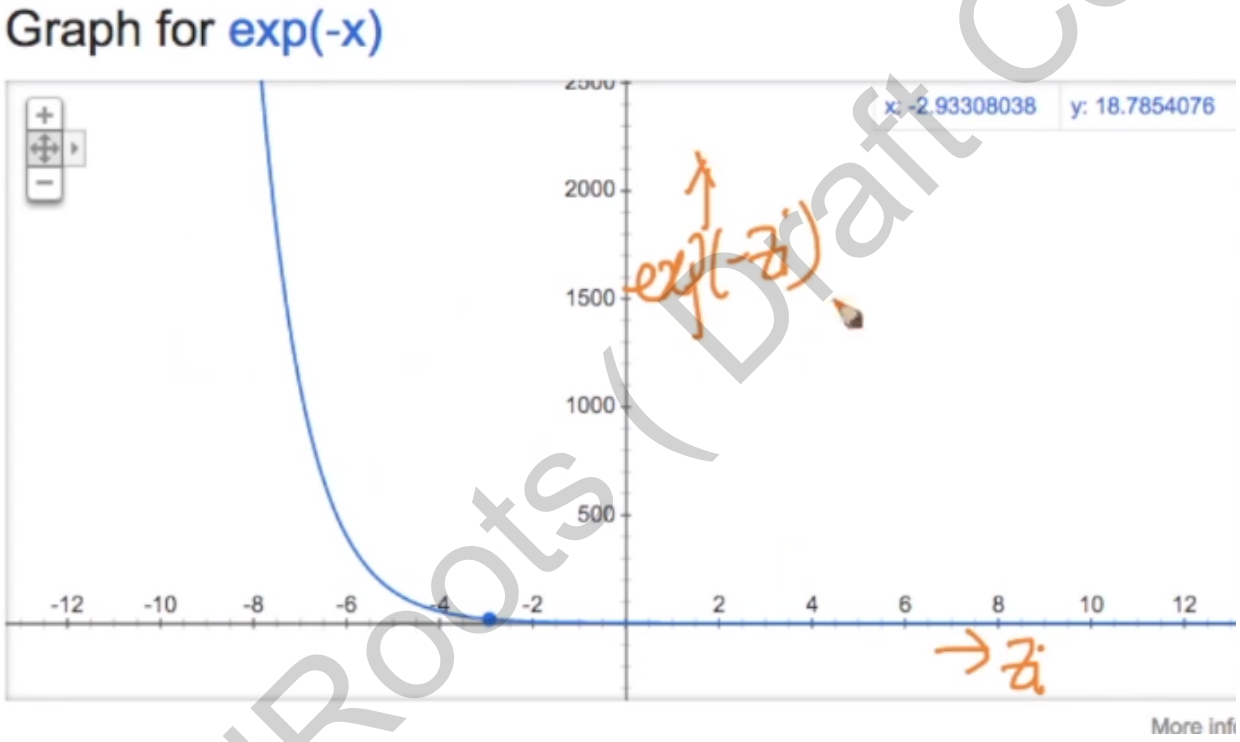
$w^* = \text{arg-min}_w \Sigma_{i=1}^n \log(1+\exp(-y_i*(w^Tx_i)))$

Let us denote **$-y_i*(w^Tx_i) = z_i$**

The optimization problem can now be written as

$w^* = \text{arg-min}_w \Sigma_{i=1}^n \log(1+\exp(-y_i*(w^Tx_i)))$

Let us have a look at the plot of $\exp(-z_i)$ below.



For whatever value of 'z_i', the value of $\exp(-z_i)$ is always greater than or equal to 0.

$\exp(-z_i) >= 0$

Now let us look into the term **$\log(1+\exp(-z_i))$**, we know that log(1)=0

And if we add any term to '1' and apply logarithm, then the value of it would be greater than or equal to 0, provided if the newly added term is non-negative.

So if **$\log(1+\delta)>=0$** (only if **$\delta>=0$**)

So ultimately **$\log(1+\exp(-zi))>=0$**

Coming back to our optimization problem
$w^* = \text{arg-min}_w \sum_{i=1}^{n} \log(1+\exp(-z_i))$

Here the minimum value of **$\sum_{i=1}^{n} \log(1+\exp(-z_i))$** is 0. This minimum value is obtained only if all the values in the sequence become 0.

The optimization problem is
$w^* = \text{arg-min}_w \sum_{i=1}^{n} \log(1+\exp(-z_i))$
From the graph of $\exp(-z_i)$, we see
As 'z_i' increases, $\exp(-z_i)$ decreases.
As 'z_i' $\rightarrow \infty$, $\exp(-z_i) \rightarrow 0$.

If 'z_i' = +ve and 'z_i' $\rightarrow \infty$, then $\exp(-z_i) \rightarrow 0$.
So gradually the term $\log(1+\exp(-z_i)) \rightarrow 0$

So the minimum value of **$\sum_{i=1}^{n} \log(1+\exp(-z_i))$** is '0' when '$z_i$' $\rightarrow \infty$ for all 'i'.
But we have assumed as $z_i = y_i*(w^Tx_i)$, $D = \{(x_i, y_i), i \rightarrow 1 \text{ to } n\}$
So here 'x_i' and 'y_i' values are constant as they are the given data points.
So the only varying term is 'w'.
We have to modify 'w' in such a way that each 'z_i' $\rightarrow +\infty$

$z_i = y_i*(w^Tx_i)$ is positive, only if the given data point (x_i, y_i) is classified correctly by the plane correctly.
$z_i = y_i*(w^Tx_i)$ is negative, only if the given data point (x_i, y_i) is misclassified.
If 'z_i' $\rightarrow +\infty$, then we reach our minima.

So the best optimal w* can be obtained only if
a) All the training data points are classified correctly
b) 'z_i' $\rightarrow +\infty$

But here is a problem when all the training data points are classified correctly. If all the training data points are classified correctly, then if there are any outliers in the training data, then model fits to the training data in such a way that even the outliers also are classified correctly, thereby

leading the model to overfit. In overfitting, classification might be perfect on the training data, but not on the test data.

But if 'z_i' $\rightarrow \infty$, then 'w' also has to be either $-\infty$ or $+\infty$, depending on the values of 'x_i' and 'y_i'. So according to the optimization problem, as we know 'w' is a vector, each component(w_i) of the vector 'w' has to be made either $-\infty$ or $+\infty$ depending on the values of 'x_i' and 'y_i' in order to arrive at a perfect classification of training data points.

So in order to get rid of the problem of overfitting, we apply a technique called **Regularization**.

L2 Regularization

The optimization problem becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^Tx_i))) + \lambda w^Tw}$$

$$\mathbf{w^Tw = \Sigma_{i=1}^{d} w_j^2 = ||w||_2^2}$$

The optimization problem now becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^Tx_i))) + \lambda\Sigma_{i=1}^{d} w_j^2}$$

The first term is the loss term and the second term is the regularization. Our job is to minimize the whole sum of both these terms.
Here if $w_j \rightarrow +\infty$ (or) $w_j \rightarrow -\infty$, then the whole sum value becomes infinity as w_j^2 -->∞. This is going to be quite opposite of what we actually need as we're actually trying to minimize the sum value.

The parameter 'λ' prevents 'w_j' from becoming $+\infty$ (or) $-\infty$. So the regularization term with the help of 'λ' is avoiding $w_j \rightarrow +\infty$ and $w_j \rightarrow -\infty$.

If $w_j \rightarrow +\infty$ (or) $w_j \rightarrow -\infty$, then $\log(1+\exp(-z_i)) = 0$, but $\mathbf{\lambda\Sigma_{i=1}^{d} w_j^2}$ becomes very large. So the regularization term and the loss term are moving in the opposite directions and are avoiding z_i's from going to infinity. But for this optimization problem, they both reach an equilibrium stage where both the terms have minimum value and here they converge to an optimal value w*.

The optimization problem is

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^T x_i))) + \lambda \Sigma_{i=1}^{d} w_j^2}$$

Here 'λ' → Hyperparameter. It's optimal value is obtained through cross validation.

If λ=0, then the model overfits. The influence of the regularization term on the model is negligible.

If 'λ' is large(say ∞), then the influence of the loss term is negligible when compared to the regularization term, and doesn't show any impact on the model, leading the model to underfitting.

33.6 - L1 Regularization and Sparsity

The optimization problem we have been solving so far is

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^Tx_i))) + \lambda||w||_2^2}$

We are using L2 Regularizer to ensure that 'z_i' don't tend to '+∞'. Otherwise 'w_i' tend to be either +∞ or -∞. Also in order to prevent the model from overfitting, we use L2 regularizer.

Here arises a question of why to use L2 regularizer and are there any alternatives for L2 regularization.

There is a popular alternative called L1 Regularization. In L1 regularization, instead of using $||w||_2^2$ for regularization, we use $||w||_1$.

So now the optimization problem becomes

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^Tx_i))) + \lambda||w||_1}$

As we are using absolute value of 'w_i' in $||w||_1$, though 'w_i' is '-ve' or '+ve', after converting into absolute format it becomes '+ve'.

Even L1 regularizer avoids $w_i \rightarrow \infty$ and $w_i \rightarrow -\infty$. L1 regularizer functions same as L2 regularizer. But it has got one advantage over L2 regularizer and it is the **Sparsity**.

Sparsity

Let us assume the optimal vector 'w' = $<w_1, w_2, w_3, \ldots, w_d>$.

A solution for Logistic Regression is said to be sparse if many w_i's are zero.

If we use L1 regularizr in Logistic Regression, all the unimportant (or) less important features in 'w' become zero.

If we have the features $f_1, f_2, f_3, \ldots, f_d$, then the corresponding components in 'w' are $w_1, w_2, w_3, \ldots, w_d$. If a feature '$f_j$' is least important, then

a) If we apply L1 regularization, 'w_j' becomes zero.
b) If we apply L2 regularization, 'w_j' becomes a small value, but not necessarily zero.

If we want all the unimportant features to become zero in our problem, we have to apply L1 regularization.

Elastic Net

Elastic Net regularization is a combination of both L1 and L2 regularization.

The optimization problem we have been working so far will become as below if we apply Elastic Net regularization.

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*(w^T x_i))) + \lambda_1||w||_1 + \lambda_2||w||_2^2}$

(loss term + L1 regularization + L2 regularization)

Here we have 2 hyperparameters 'λ_1' and 'λ_2'. These two hyperparameters can be found using cross-validation.

Elastic-Net results in sparse solutions as it contains 'L1' regularization while avoiding some of the disadvantages of 'L1' regularization.

On the other hand, 'L2' regularization tends to typically result in better performance, but no sparsity. Now the elastic net tries to combine the advantages of both these methods while minimizing their disadvantages.

It is preferred to use ElasticNet, but it takes more time to train with ElasticNet regularization as compared to 'L1' and 'L2'.

If $\lambda_1=0 \rightarrow$ ElasticNet reduces to L2 regularization.
If $\lambda_2=0 \rightarrow$ ElasticNet reduces to L1 regularization.

Note

L1 regularization can be used in any ML model to introduce sparsity. We use L1 regularization to zero out the weights of the least important features.

Both L1 and L2 regularizations are used to avoid overfitting. But along with avoiding overfitting, if we have a requirement of low latency, we have to prefer L1 regularization.

33.7 - Probabilistic Interpretation: Gaussian Naive Bayes

In Naive Bayes,

a) If features are real valued, then we can assume that the real valued features have Gaussian Distribution.
b) If y_i = +1 or 0, then we can think of our class label as a Bernoulli Random variable.

If these two conditions are satisfied, then we can easily derive the whole Logistic Regression.

In probabilistic interpretation, using Gaussian Naive Bayes,

Logistic Regression = Gaussian Naive Bayes + Bernoulli Distribution

Assumptions made in Probabilistic Interpretation of Logistic Regression

a) The response variable 'Y' is boolean, governed by a Bernoulli distribution, with parameter $\pi = P(Y=1)$
b) All the features 'X_i'(i = 1 to d) are continuous random variables.
c) For each 'X_i', $P(X_i|Y=y_k)$ is a Gaussian distribution of form $N(\mu_{ik}, \sigma_i)$.
d) For all i≠j, 'X_i' and 'X_j' are conditionally independent given 'Y'.

In the geometric interpretation of Logistic Regression, the optimization problem is

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*((w^Tx_i)/||w||))) + Regularization}$$

Here $y_i \in \{-1, +1\}$

In the probabilistic interpretation of Logistic Regression, the optimization problem is

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} -y_i logP_i - (1-y_i)log(1-P_i) + Regularization}$$

Case 1

When y_i = +ve, then

In geometric interpretation, y_i = +1

In probabilistic interpretation, y_i = +1

In geometric interpretation, if 'w' is not a unit vector, then the optimization problem becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n \log(1+\exp(-(w^T x_i)/||w||))}$$

In probabilistic interpretation, if 'w' is not a unit vector, then the optimization problem becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n -\log(1/(1+\exp(-(w^T x_i)/||w||)))}$$
$$\mathbf{= arg\text{-}min_w \Sigma_{i=1}^n \log(1+\exp(-(w^T x_i)/||w||))}$$

Case 2

When y_i = -ve, then

In geometric interpretation, $y_i = -1$

In probabilistic interpretation, $y_i = 0$

In geometric interpretation, if 'w' is not a unit vector, then the optimization problem becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n \log(1+\exp((w^T x_i)/||w||))}$$ (because 'y_i'=-1)

In probabilistic interpretation, if 'w' is not a unit vector, then the optimization problem becomes

$$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^n -\log(1-(1/(1+\exp((w^T x_i)/||w||))))}$$
$$\mathbf{= arg\text{-}min_w \Sigma_{i=1}^n -\log(\exp((w^T x_i)/||w||)/(1+\exp((w^T x_i)/||w||)))}$$
$$\mathbf{= arg\text{-}min_w \Sigma_{i=1}^n \log(1+\exp((w^T x_i)/||w||))}$$

Note

Once if we have the same optimization which can be arrived at using either the probabilistic approach or geometric approach, we can solve the optimization problem using stochastic gradient descent.

33.8 - Loss Minimization Interpretation

The optimization problem is

$\mathbf{w^* = arg\text{-}min_w \Sigma_{i=1}^{n} log(1+exp(-y_i*((w^Tx_i)/||w||)))}$

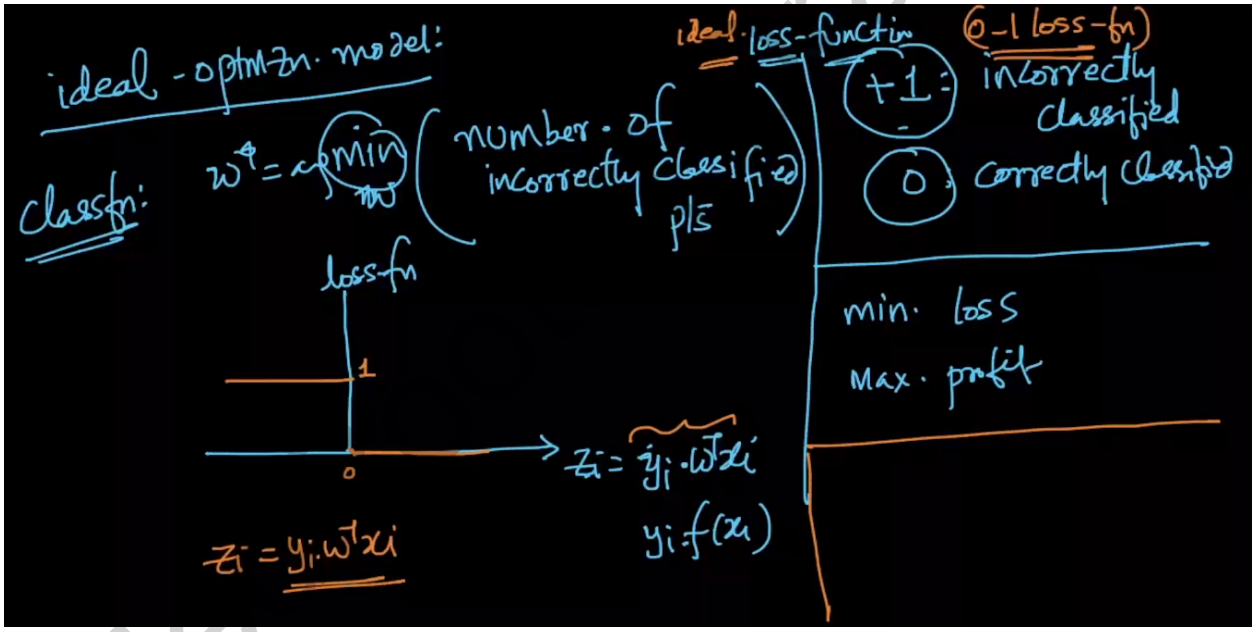
If 'w' is a unit vector, then ||w|| = 1.

$\mathbf{z_i = y_i*(w^Tx_i)/||w|| = y_i * f(x_i)}$ (where $f(x_i) = (w^Tx_i)/||w||$)

Let us go with building an ideal optimization model. Let us assume a function which returns '+1' if a data point passed through it is incorrectly classified and '0' if a data point passed through it is correctly classified. This function is called **Loss Function**.

So now the optimization problem would be

$\mathbf{w^* = arg\text{-}min_w}$ **(Number of Incorrectly classified points)**



As we know that

If 'z_i' is +ve, then the data point (x_i, y_i) is classified correctly.

If 'z_i' is -ve, then the data point (x_i, y_i) is classified incorrectly.

So we need a loss function such that

Loss-function = 0, if z_i>0

= 1, if z_i<0

Such a loss function is called **0-1 loss function**.

0-1 loss(z_i) = **0, if $z_i>0$**
= **1, if $z_i<0$**

From the loss minimization perspective, the ideal 'w' is obtained by

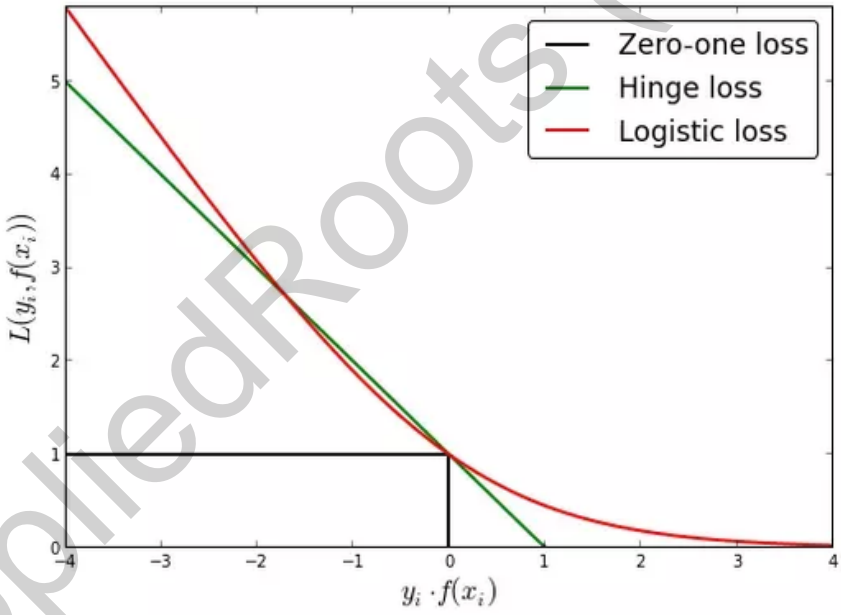
$w^* = \text{arg-min}_w \Sigma_{i=1}^{n}$ 0-1 loss(z_i)

To solve optimization problems in ML, we'll use differentiation in calculus.

But a function is differentiable only if it is continuous. Similarly, a function is non-differentiable if it is not continuous. From this problem, we have the 0-1 loss function being defined in the intervals (-∞, 0) and (0, ∞).

But it is not defined for $z_i=0$. Hence we couldn't say it is differentiable, as it is not continuous at $z_i=0$. Even though the 0-1 loss function is ideal, we could not solve the problem.

In optimization problems, if we are not able to solve them, we can approximate them. One such approximation for this problem is **Logistic Loss**.



From the above graph, we see that

If z_i is '+ve', then (0-1) loss = 0

As $z_i \rightarrow \infty$, Logistic Loss $\rightarrow$ 0.

If z_i is '-ve', then (0-1) loss = 1
As $z_i \rightarrow -\infty$, Logistic Loss $\rightarrow$ very large value.

In this problem, we are using Logistic Loss as an approximation over 0-1 loss. As we are using Logistic Loss as an approximation, we get the Logistic Regression model.

Similarly, if we use Hinge Loss as an approximation, we get the Support Vector Machine model.

33.9 - Hyperparameter Seach: Grid Search and Random Search

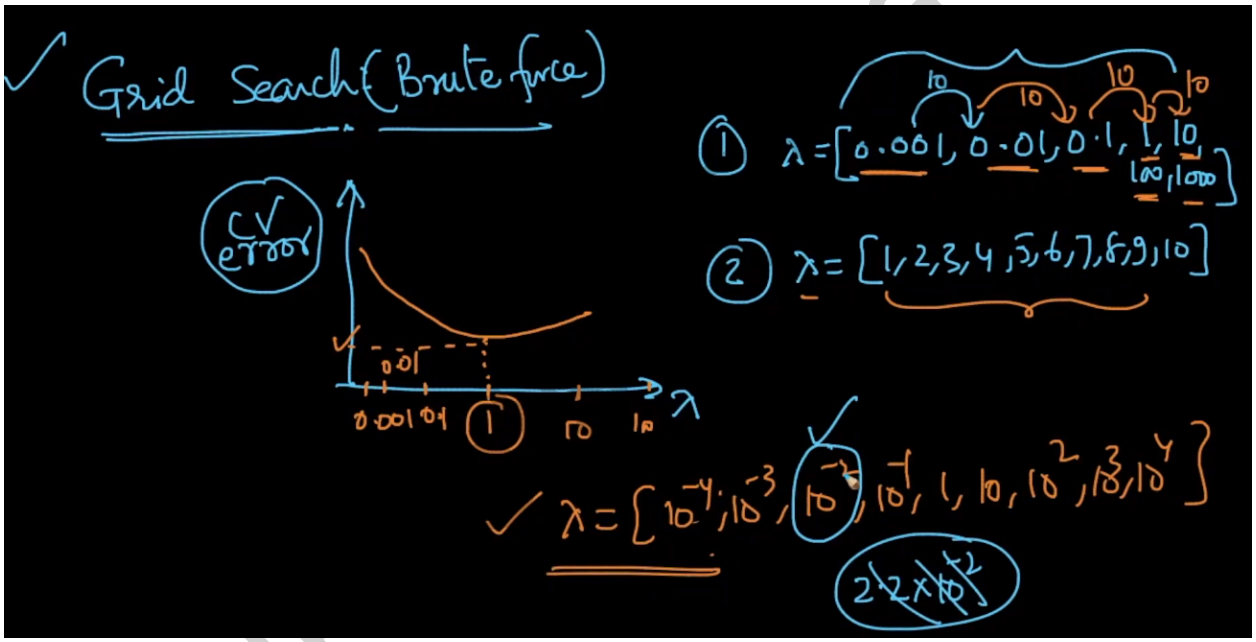
So far we have seen the hyperparameter 'λ' in the formulation of Logistic Regression.

If λ = 0 → Overfitting

If λ = ∞ → Underfitting

One job here is to find the optimal value of 'λ'.

Grid Search Cross-Validation

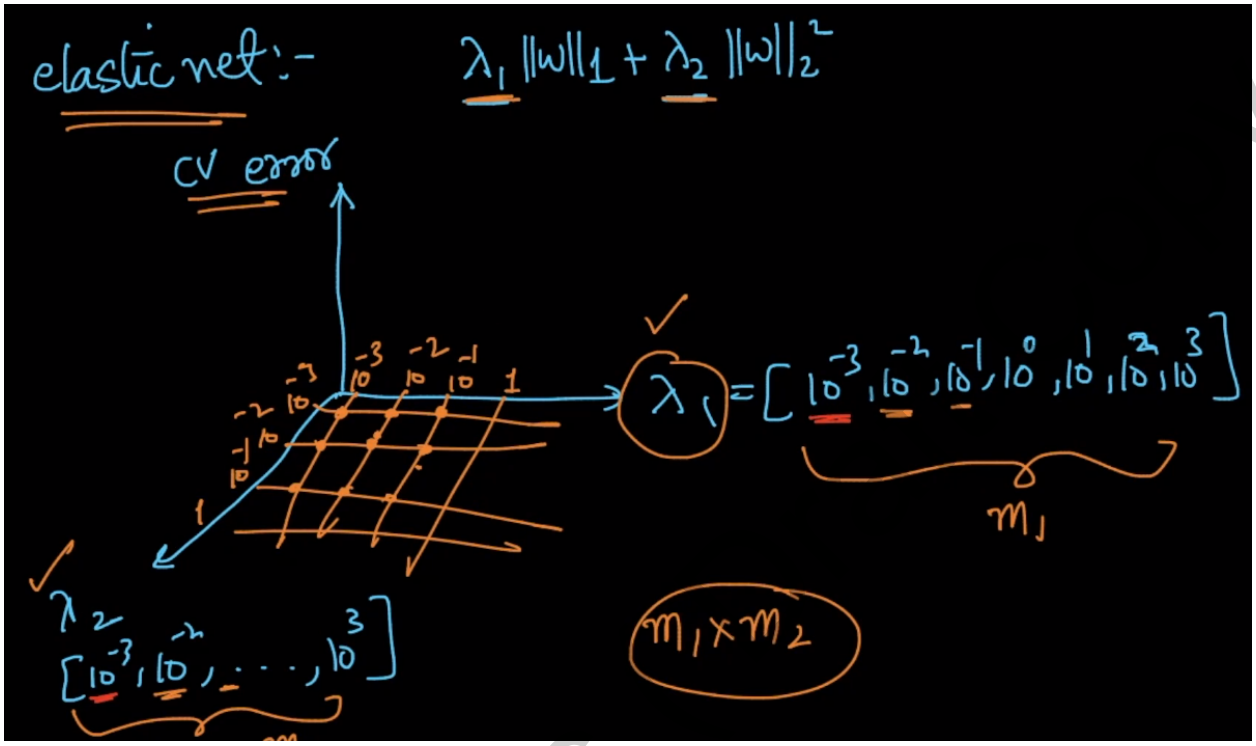


In Logistic Regresion with either 'L_1' or 'L_2' regularization, 'λ' is the only hyperparameter we have to tune. In K-NN, the hyperparameter 'K' takes only integers. Similarly in Logistic Regression, 'λ' takes only the real values.

So we have to try different values of 'λ' and compute the cross validation error for each value of 'λ', and whichever value of 'λ' yields the least cross validation error, that will be the optimal value of 'λ'.

Let us assume if we are using around 'm' different values for tuning of 'λ', then we have to train the model 'm' times.

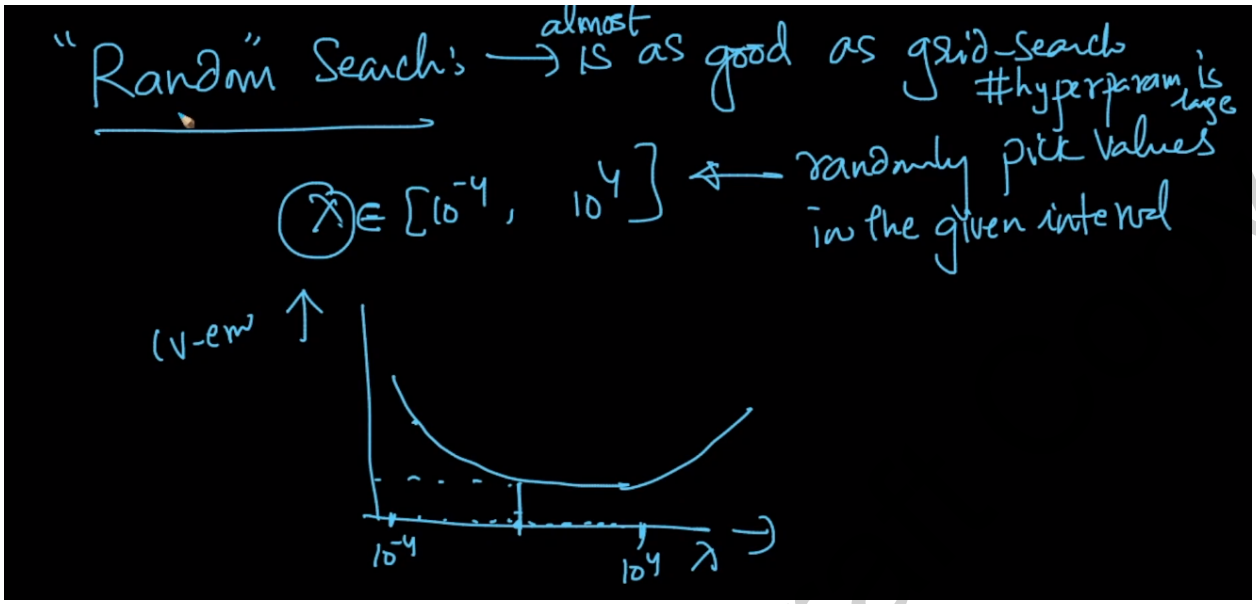
If we apply Elasticnet Regularization(ie.,both 'L_1' & 'L_2' regularization), then we have to find out the optimal values for the two hyperparameters 'λ_1' and 'λ_2'.



Let the count of values used for tuning 'λ_1' be 'm_1', and the count of values used for tuning 'λ_2' be 'm_2', then the total number of combinations is **$m_1 * m_2$**.

As the number of hyperparameters increase, the number of times the model needs to get trained increases exponentially. But in Logistic Regression, at the max, we could have 2 hyperparameters, whereas in neural networks, we have multiple hyperparameters to tun, and Grid Search Cross-Validation is not a good choice. In such a case, we have an alternative technique to go with and it is the **Random Search Cross Validation**.

Random Search Cross Validation



If we have a hyperparameter 'λ' and if an interval is given for 'λ', then this technique picks random values in the given interval for 'λ' and plots the cross validation errors associated with those random values of 'λ'. Whichever value of 'λ' gives the least cross validation error, that is chosen as the optimal value.

Random Search cross validation is as good as Grid Search cross validation, and is faster than Grid Search especially when the number of hyperparameters is large.

These two techniques can not only be used for Logistic Regression, but also can be used for any other ML algorithm like K-NN, Naive Bayes, etc.

Scikit-Learn Documentation Links

Grid Search CV :
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
Random Search CV:
https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

33.10 - Column Standardization

Let us assume we have a dataset with ‘d’ features and ‘n’ data points.
Here $x_i \in R^d$
If we apply Feature/Column Standardization, then each value ‘x_{ij}’ becomes
$\mathbf{x_{ij}^l = (x_{ij} - \mu_j)/\sigma_j}$
where i → 1 to ‘n’, j → 1 to ‘d’

Same as K-NN, even in Logistic Regression, it is important and mandatory to perform column/feature standardization. It is because in K-NN, we compute the distances between the data points. Similarly, in Logistic Regression, we compute the distances of the data points from the hyperplane separating different classes.

As the distances could easily get impacted by the features being present on different scales, the feature/column standardization is mandatory. Moreover standardization of features lead to faster convergence.

33.11 - Feature Importance and Model Interpretability

Let us assume we have features f_1, f_2, f_3, ….., f_d in our dataset. We shall compute the weight vector and let the components be w_1, w_2, w_3, …., w_d.

Let us make an assumption that all the features in our dataset are conditionally independent of each other(same like in Naive Bayes because Logistic Regression = Gaussian Naive Bayes + Bernoullli Distribution)

If we consider this assumption to be True, then we can make an interesting conclusion. That is the feature importance could be obtained using the values of **w_j's** only if the assumption is True. If the assumption is False, then the conclusion also becomes False.

In K-NN, the feature importance is obtained using the feature selection techniques like Forward Feature Selection, Backward Feature Elimination, etc.

In Naive Bayes, the feature importance is obtained using the values $P(w_i|y=1)$.

In Logistic Regression, the feature importance could be obtained by **w_j's**.

How to get the feature importance using 'w_j'?

For a feature 'f_j', its corresponding weight component is 'w_j'.
$|w_j|$ → Absolute value of weight corresponding to 'f_j'.
If **$|w_j|$** is large(whether it is +ve or -ve), its contribution to **(w^Tx)** is large.

Case 1

If 'w_j' is +ve and large, it shows a huge positive impact on $\Sigma_{i=1}^{d} w_j.x_{qj} = w^Tx_q$, and the value of $P(y_q=+1)$ increases.

Case 2

If 'w_j' is -ve and large, it shows a huge negative impact on $\Sigma_{i=1}^{d} w_j.x_{qj} = w^Tx_q$, and the value of $P(y_q=-1)$ increases.

So we can determine the important features in Logistic Regression model, on the basis of the $|w_j|$ values.

Example

Let us assume we have to predict the gender of a given person, and out of all the features we have in our dataset, 'hair-length' is one among them. Let us denote the coefficient of 'Hair-Length' as 'w_{HL}'. Let us denote 'Male' gender as class label '1', and the female gender as the class label '-1'.

If $|w_{HL}|$ is large, as we know from common sense point that women tend to have longer hair then men. So here as women have longer hair, 'f_{HL}' tend to be more and more negative(it means large value, but negative), and the class label for this data point will be predicted as '-1', as the value of $P(y_q = -1)$ increases with large negative value of 'w_{HL}'.

Model Interpretability

In case of prediction, it would be sufficient if we just predict the value of 'y_q' either as -1 or +1, for a given data point 'x_q'.

But in the case of interpretability, we also should be able to give the reasons for the predictions.

It is hard to interpret the feature importance geometrically, but is easier and more intuitive to understand it from an algebraic perspective.

33.12 - Collinearity of Features

So far we have seen that $|w_j|$ is used in determining the feature importance only if all the features are independent of each other.

If we have two features 'f_i' and 'f_j', and if one feature can be expressed as a linear function of the other feature (say $\mathbf{f_i = \alpha f_j + \beta}$), then we can say that both 'f_i' and 'f_j' are collinear.

If we have the features f_1, f_2, f_3, ….., f_d and if they all are linearly related such as $f_1 = \alpha_1 + \alpha_2 f_2 + \alpha_3 f_3 + \ldots. + \alpha_d f_d$, then we can say that f_1, f_2, f_3, ….., f_d are multicollinear.

Why is $|w_j|$ not useful in determining feature importance if features are collinear?

Let us assume we have a dataset $D = \{x_i, y_i\}_{i=1}^{n}$

Let the optimal $w^* = <1,2,3>$

The corresponding $x_q = <x_{q1}, x_{q2}, x_{q3}>$

So if we compute '$w^{*T}x_q$',

$w^{*T}x_q = x_{q1} + 2x_{q2} + 3x_{q3}$ ------ (1)

Let's say $f_2 = 1.5f_1$, then

$x_{q2} = 1.5x_{q1}$ ------ (2)

We can say 'f_1' and 'f_2' are collinear. If we substitute (2) in (1), we get

$w^{*T}x_q = x_{q1} + 2(1.5x_{q1}) + 3x_{q3} = x_{q1} + 3x_{q1} + 3x_{q3} = 4x_{q1} + 3x_{q3}$ ------ (3)

So now the component coefficients have become <4,0,3>.

But the original vector coefficients of $w^{*T}x_q$ in (1) are <1,2,3>, but due to collinearity, we got the coefficients as <4,0,3> in (3).

Here these two are entire different weight vectors, but still they represent the same classifier. Also these two vectors ultimately give the same value of '$w^{*T}x_q$' for a data point 'x_q', because the two features 'f_1' and 'f_2' are collinear.

Let us take those two vectors again.

$w^* = <1,2,3>$ ($f_1 = 1$, $f_2 = 2$, $f_3 = 3$)

$\hat{w} = <4,0,3>$ ($f_1 = 4$, $f_2 = 0$, $f_3 = 3$)

From ‘w*’, we come to a conclusion that ‘f_3’ is the most important feature, whereas from ‘$\hat{w}$’, we come to a conclusion that ‘f_1’ is the most important feature, and ‘f_2’ is useless.

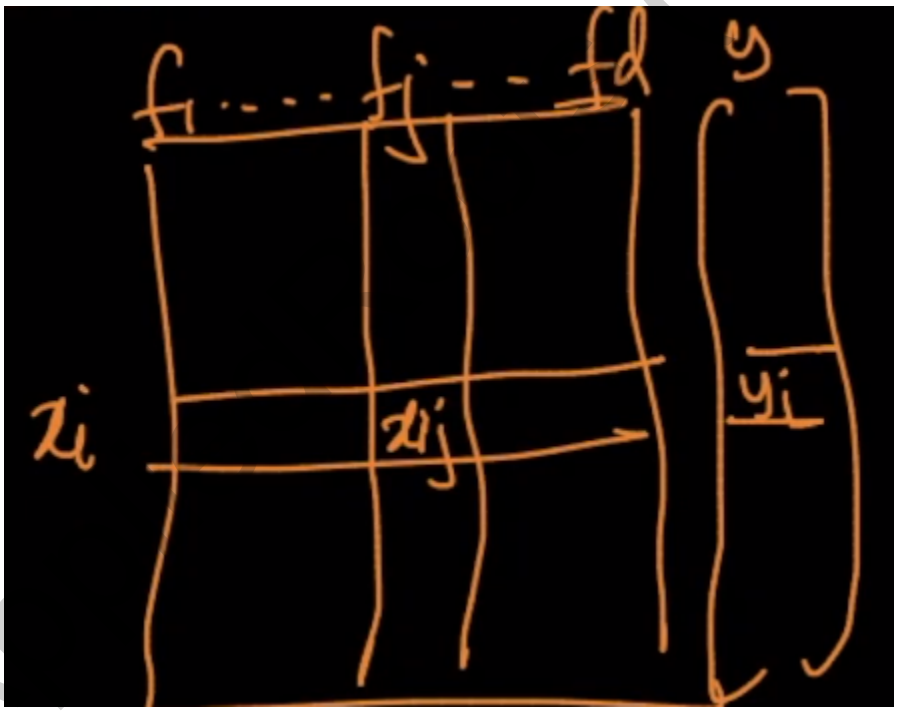
The conclusion drawn from both these vectors are quite opposite. There are more chances of misinterpretations because of collinearity among the features.

So if the features are collinear, the weight vector can change arbitrarily. Hence $|w_j|$ cannot be used in determining feature importance when any of the given features are collinear.

Before we use $|w_j|$ for determining the feature importance, we have to determine if there exists multi-collinearity between the features. The best technique to determine if the features are multi-collinear is **Pertubation** technique.

Pertubation

Let us assume we have a dataset ‘X’ of ‘n’ data points and each data point has ‘d’ dimensions.



We have apply Standardization/Normlization (as per our problem requirement) on the features.

We have to train the Logistic Regression model on the data matrix and obtain the weight vector ‘w-bar’.

Now we have to take each value ‘x_{ij}’ and pertubate them. Here perturbation in the sense, adding a small noise. Let the noise be denoted as ‘$\in$’. So train the logistic regression model using the perturbation data, and obtain the weight vector ‘w-tilda’.

w-bar = <$w\text{-}bar_1$, $w\text{-}bar_2$, $w\text{-}bar_3$, ……, $w\text{-}bar_d$>

w-tilda = <$w\text{-}tilda_1$, $w\text{-}tilda_2$, $w\text{-}tilda_3$, ……, $w\text{-}tilda_d$>

We now have to check how the components of these two vectors differ. If the differences between the components of ‘w-bar’ and ‘w-tilda’ are huge, then we can say that our features are collinear. Then we can’t use $|w_j|$ to determine the feature importance.

If the differences between the components of ‘w-bar’ and ‘w-tilda’ are small, then we can neglect the collinearity issue.

If we have the problem of collinearity, then as we could not use $|w_j|$ to determine feature importance, we have to go for generic techniques like Forward Feature Selection to determine the feature importance.

But before using $|w_j|$ for determining feature importance, we always have to perform collinearity check.

33.13 - Train and Runtime Space and Train Complexities

The problem of optimization that has to be solved in Logistic Regression is

$w^* = \text{arg-min}_w$ (Logistic Loss + Regularization)

This problem is going to be solved using Stochastic Gradient Descent algorithm.

For this problem, let 'D' be the given training dataset.

n → number of points in the training data

d → number of dimensions

For Training Phase,

Time Complexity = O(nd)

For Runtime,

Space Complexity = O(d) (It is because, we compute the vector 'w*' and store each component of 'w' which is d-dimensional)

Time Complexity = O(d) (It is because, we compute $\mathbf{\Sigma_{j=1}^{d}\ w_j x_{qj}}$ which has 'd' terms in th sequence)

Note

- If 'd' is small, then Logistic Regression is good for low latency applications.
- If 'd' is large, then we have to apply L1 regularization that introduces sparsity (as it makes w_j's corresponding to least important features equal to 0).
- In case, if we still have more number of computations that could not be performed by the system, then evn after having sparsity, we have to keep increasing the value of 'λ', which increases the sparsity and makes more number of weight components equal to 0 and thereby reducing the number of computations.
- But increasing/decreasing 'λ' by a large amount leads to underfitting/overfitting. Hence we need to modify 'λ' in such a way that it keeps a balance between Bias, Variance and Latency time.

33.14 - Real-World Cases

1) **Decision Surface**

We have a linear decision surface in Logistic Regression. In 2-D space, it is called a line, in 3-D it is called a plane and in n-D space it is called a hyperplane.

2) **Assumption**

Logistic Regression works on the assumption that the given data is perfectly/almost linearly separable.

3) **Feature Importance**

The feature importance in Logistic Regression is obtained by the values of $|w_j|$, if the features are not collinear. If collinearity exists among the features, then we have to choose techniques like Forward Feature Selection, Backward Feature Elimination, etc.

4) **Imbalanced Data**

We can balance the dataset either using upsampling (or) downsampling (or) on the basis of class weights.

5) **Impact of Outliers**

The impact of the outliers is very low as we have the sigmoid function which reduces the magnitude of signed distance. The impact of the outliers is very low, but not completely vanished.

Procedure for outlier removal that can be followed

a) For every point 'x_i' in D_{Train}, compute the distance of it to the hyperplane. (ie., you have to compute $w^T x_i$)

b) Remove those points from D_{Train}, which are far away from the hyperplane, and keep the remaining points. Let the dataset with the remaining points be $D_{Train}^{\prime}$

c) Compute the optimal 'w' for $D_{Train}^{\prime}$ as the final solution.

6) **Missing Values**

We can use the standard imputation techniques(that were discussed in the course) to handle the missing values in the data.

7) **Multi-class problems**

We can go with the one-vs-rest approach in case of a multi-class classification problem. There are also extensions of Logistic Regression to multi-class classification. They are Maximum Entropy model, Softmax classifier and Multinomial Logistic Regression.

8) **Similarity/Distance Matrix**

There is an extension of Logistic Regression called Kernel Logistic Regression. It takes similarity matrices as input, whereas the normal Logistic Regression, doesn't accept the similarity matrices as input.

9) **Best and Worst Cases of Logistic Regression**

Best Case

a) If our data is almost/perfectly linearly separable.

b) If we have low latency requirement. (ie., especially when L1 regularization is applied)

c) It is very fast to train.

Worst Case

When our data is not linearly separable.

10) **High Dimensionality**

a) When the dimensionality is high, the chances for our data to be linearly separable are also high.

b) Logistic Regression works well when the dimensionality is large.

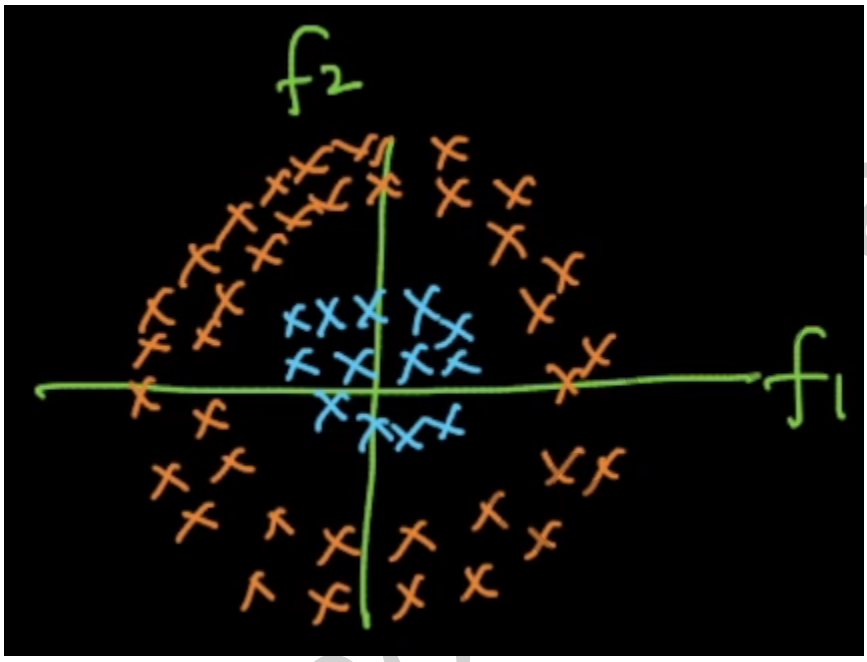
c) If we want a low latency system in presence of high dimensionality, then we have to apply L1 regularization.

33.15 Non Linearly Separable Data and Feature Engineering

Logistic Regression in general works on the assumption that the given data is linearly separable.

Example 1

Let us assume that have the given data as shown below.

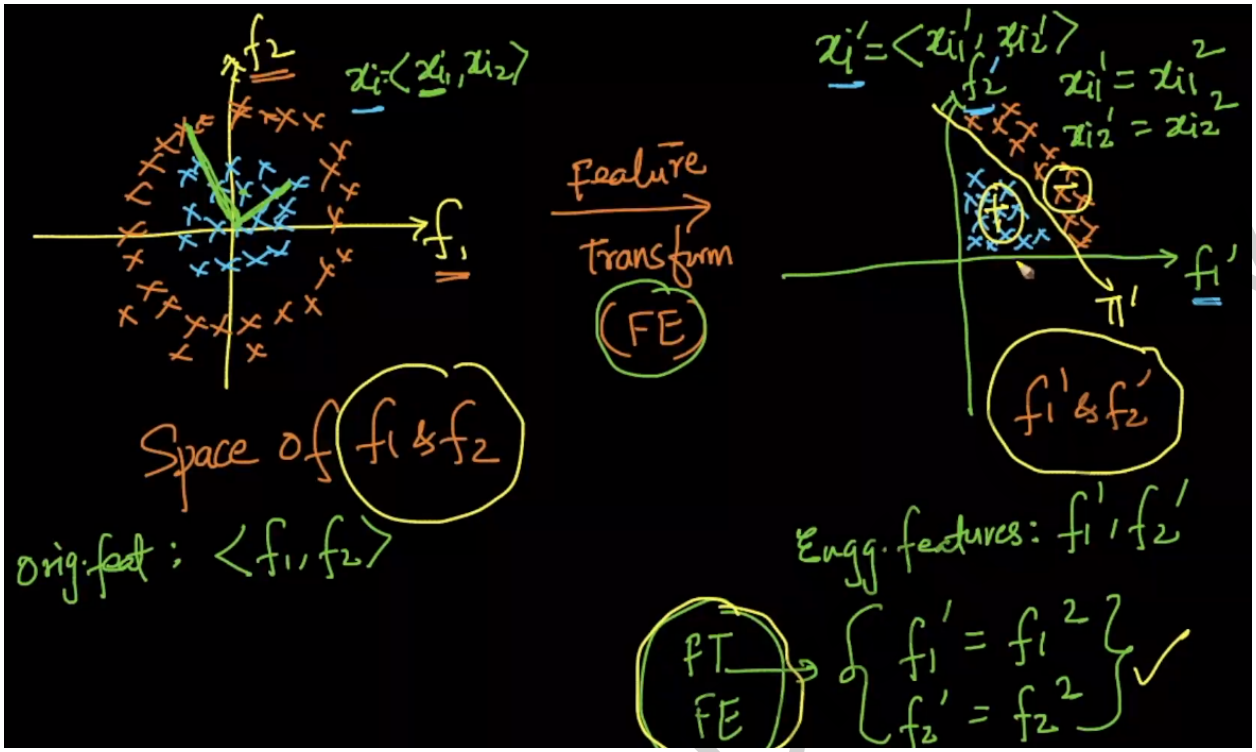


Here the data isn't linearly separable. In such cases, we cannot draw a line/plane/hyperplane in the space of 'f_1' and 'f_2' to separate the positive and the negative points, as it doesn't work.

Moreover, at the same time, we also couldn't conclude that Logistic Regression is not going to work in this context. We can apply Logistic Regression by following some ways.

In order to make the data suitable for applying Logistic Regression, we have to perform Feature Engineering (ie., applying some transformations to the existing features).

Let us have a look at the below geometric representation of the same data.



Through Feature Engineering we are converting the features 'f_1' and 'f_2' as $\mathbf{f1^{l}=f_1^2}$ and $\mathbf{f_2^{l} = f_2^2}$. After feature engineering, the data has become linearly separable. So we can now apply Logistic Regression.

How do we know which transform to apply?

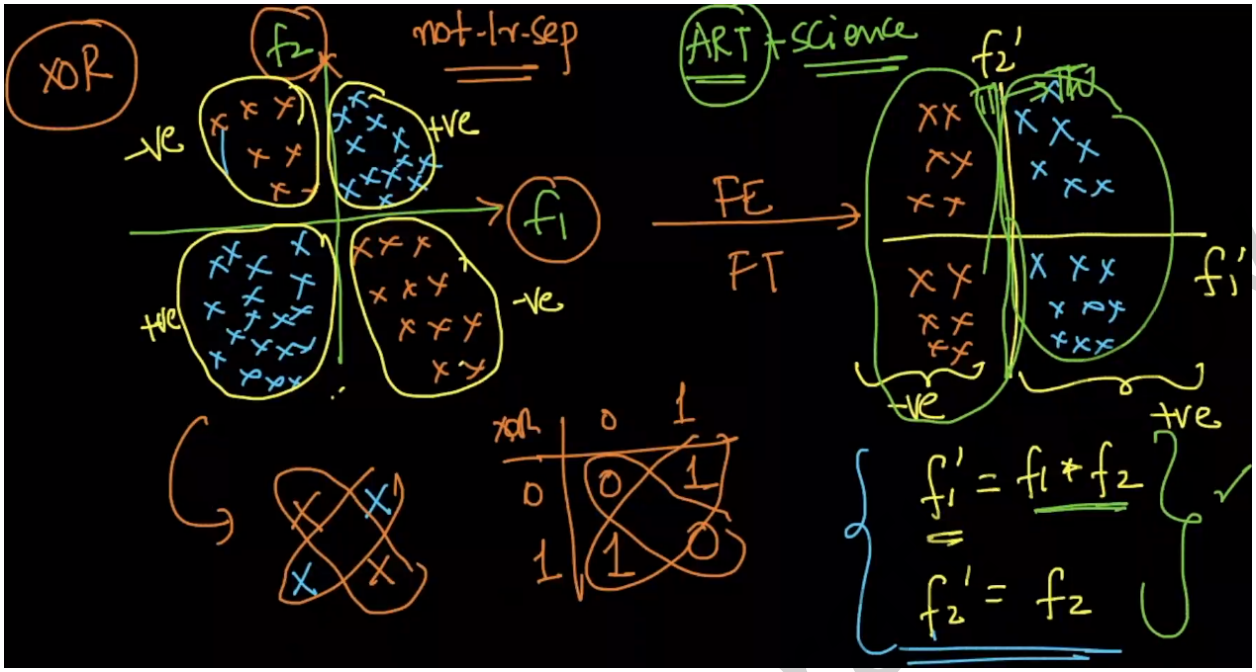
We see that the given data is in the form of a circle. As we have 'f_1' and 'f_2' as features, we should consider

$f_1^2 + f_2^2 = r^2$ ------ (1)

Here equation (1) is not linear in 'f_1' and 'f_2' but is linear in 'f_1^2' and 'f_2^2'. So if $\mathbf{f_1^{l} = f_1^2}$ and $\mathbf{f_2^{l} = f_2^2}$, then we can say equation (1) is linear in 'f_1^{l}' and 'f_2^{l}'.

Example 2

Let us look at another dataset whose points are present in the 2D space as shown below



Here left side figure shows us the actual position of the points in the 2D space. We need to apply feature transforms such that we could separate the classes using a line.

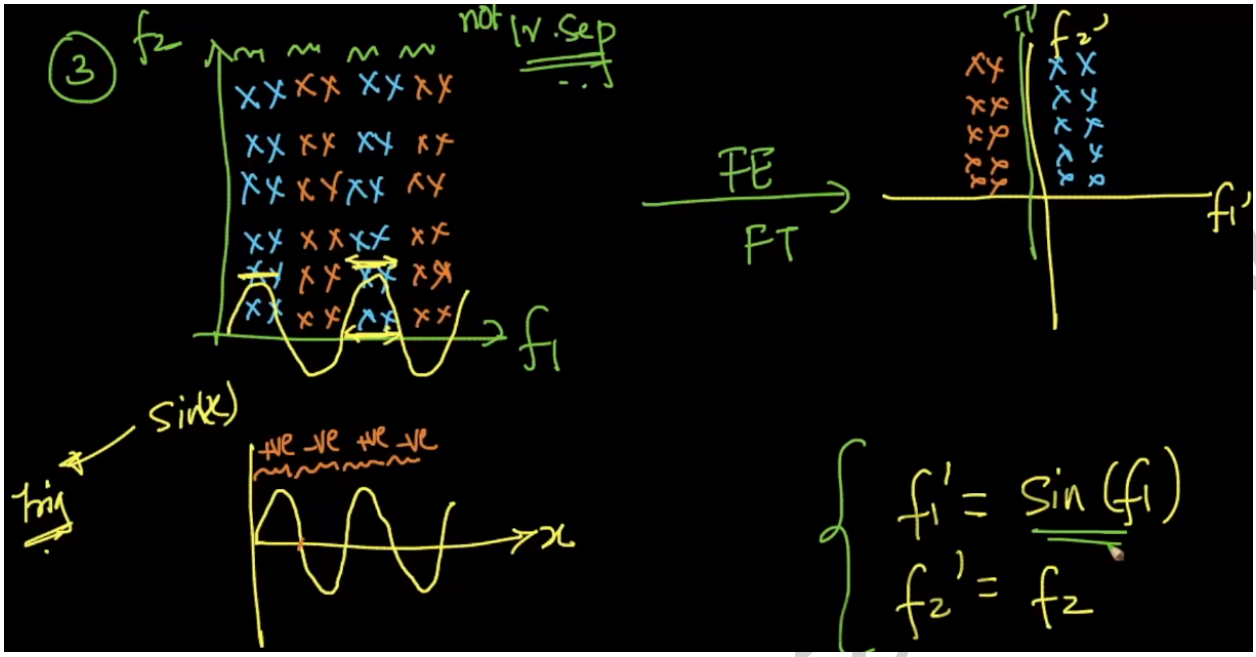
When we look at the points in the left side figure, we can see the figure is in XOR form. So the appropriate transformation that we can apply to make the data linearly separable as shown in the right side figure is

$\mathbf{f_1' = f_1 * f_2}$

$\mathbf{f_2' = f_2}$

Example 3

Let us look at another example in the below figure.



Here the input looks in the form of a sinusoidal wave. So the appropriate transformation that has to be applied in order to transform the data into a linearly separable format is

$f_1^{|} = \sin(f_1)$

$f_2^{|} = f_2$

Typical Transforms for Real-Valued Functions

1) **Polynomial Feature Transforms**
   Examples: f_1*f_2, f_1^2, f_2^2, f_2^3, f_1^3,...., $f_1^2f_2$,....
2) **Trigonometric Feature Transforms**
   Examples: $\sin(f_1)$, $\cos(f_1)$, $\sin(f_1)*\cos(f_2)$, $(\sin(f_1))^2$, ….
3) **Boolean Feature Transforms**
   Examples: OR, AND, XOR
4) **Mathematical Feature Transforms**
   Examples: $\log(f_1)$, $\exp(f_1)$, ……

Note: Regarding the video lecture 33.16, we aren't giving any notes, as it is purely a discussion on the code samples. We are providing you with the link to download the ipython notebook, and if you still have any queries, please feel free to post them in the comments section.

Link:https://drive.google.com/file/d/1WcVTklMZBMu9VTCIWeupOK0r2aYbHk8p/view

33.17 - Extensions to Logistic Regression: Generalized Linear Models (GLM)

There is an extension to Logistic Regression called Generalized Linear Model (GLM).
In probabilistic interpretation,

Logistic Regression = Gaussian Naive Bayes + Bernoulli Distribution

a) In Logistic Regression, if we change the distribution of random variable from Bernoulli to Multinomial, then the resulting Logistic Regression model is called **Multinomial Logistic Regression**. This is useful when we have a multi-class classification problem to solve. It is also known as the soft-max classifier.
b) In Logistic regression, if we assume that $P(y|X) \sim N(\mu, \sigma^2)$, then we call the resulting algorithm a **Linear Regression**, and it is a regression technique. (Here $y_i \in R$)
c) If the output variables y_i's are poison distributed, then we get Poisson Regression. Poisson Regression is used to measure the counts.

Reference Link

Generalized Linear Models:
http://cs229.stanford.edu/notes2020spring/cs229-notes1.pdf (Part 3)