Machine Learning.

Machine Learning : → Supervised Learning
→ Unsupervised learning → clustering

Supervised → Regression
→ Classification

Supervised (output is known)

Regression → 1. Linear Regression
2. Polynomial Regression
3. SVR
4. Decision Tree
5. Random Forest
6. Xgboost , 7. KNN

Classification: → 1. Logistic Regression
2. SVM
3. Decision Tree.
4. Random Forest
5. Naive Bayes
6. KNN

Unsupervised learning: (output is unknown)

clustering → DBScan, kmeans,
Hierarchical,
Silhouette scoring

Supervised Learning

Eg. Independent feature → dependent features

Degree	Experience	Salary
B.Tech	7	50k
PhD	2	70k
–	–	

o/p = continuous → Regression problem

Eg. Independent feat. → dependent

No. of play hrs	No. of study hrs	Pass/Fail feat.

o/p = categorical = classification problem

- Flight Price Prediction → Regression problem
- Algerian Fire Forest → classification problem
- Predict Air Quality Index → Regression Problem
- Rain Prediction → classification Problem
- Buying day of a person → classification problem

Note: Independent features is basically input features.

Simple Linear Regression.

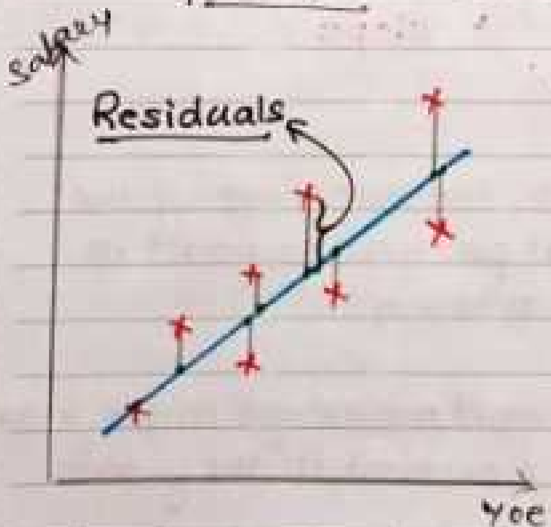
(one independent feature and one dependent feature)

Eg. aim: to create a model, which takes input as height and predict weight.

dataset: height, weight.

Eg. aim: Based on the no. of rooms, predict price.

Eg. model: year of experience and salary
predict: salary band on i/p year.

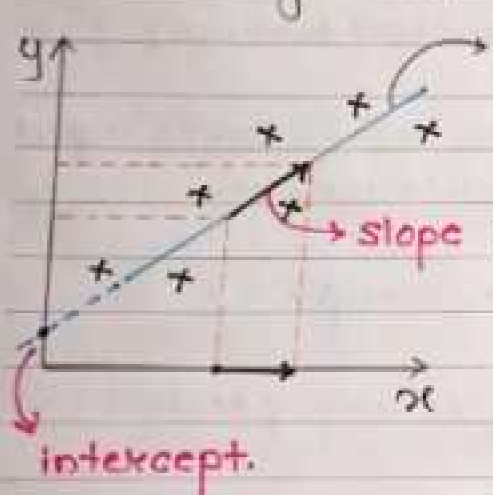


× → datapoint (original)
• → predicted data point.

• difference between real points and predicted points is called residuals or Errors.

- Based on the training dataset, it finds the best fit line in such a way that the sum of difference between real points and predicted should be minimum.

First, we need to understand why are creating a straight line?



Best best fit line is nothing but a equation of straight line

$$
\left.
\begin{aligned}
y &= mx + c \\
y &= \beta_0 + \beta_1 x \\
h_\theta(x) &= \theta_0 + \theta_1 x
\end{aligned}
\right\}
\text{ all are eq}^{n}\text{. of line}
$$

$$h_\theta(x) = \theta_0 + \theta_1 x$$

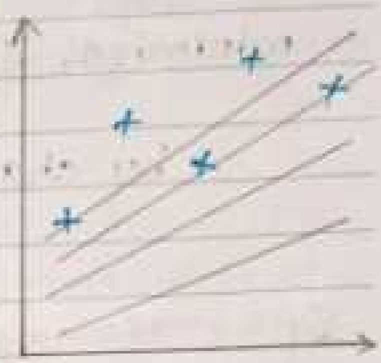
θ_0 → intercept

$\theta_1 x$ → slope

$\boxed{\text{intercept}}$: when $x=0$, the line meeting the y-axis, that particular point is known as intercept.

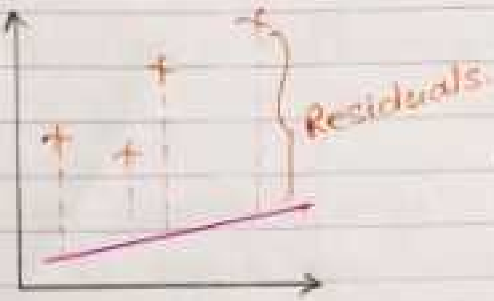
$\boxed{\theta_1 : \text{Slope:}}$ with the unit movement in the x-axis what is the movement in the y-axis.

- By changing θ_0 and θ_1, best fit line will be change.



θ_0, θ_1: changing the values, will change the line.

after some iterations, we will get a best fit line which is known as training of the model.



we need to minimize this errors using equation called as cost function.

Cost Function:

$$J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\Big(\underbrace{h_\theta(x)^{(i)}}_{\text{predicted}} - \underbrace{y^{(i)}}_{\text{actual}}\Big)^2$$

for easy calculations (derivatives)

mean square error. (one of the cost function)

- we need to minimize cost function to get the best fit line.
- dividing by m to get the average (mean).

final aim:

minimize, $J(\theta_0,\theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x)^{(i)} - y^{(i)}\right)^2$

· by changing the values of θ_0 and θ_1.

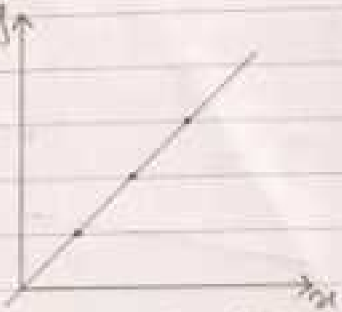
Let us consider,

$\theta_0 = 0$

$h_\theta(x) = \theta_1 x$

Eg. Training dataset.

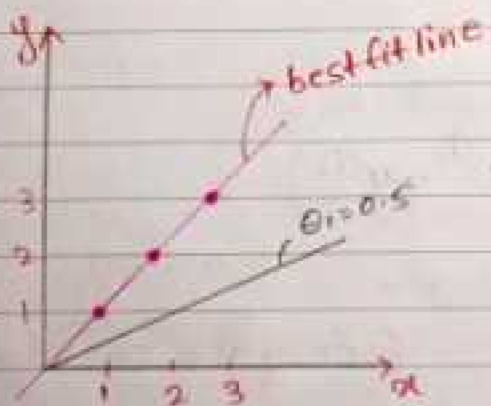
x	y
1	1
2	2
3	3



$h_\theta(x) = \theta_0 + \theta_1 x$

Now, let's assume $\theta_1 = 1$

$\therefore\ h_\theta(x) = x$



$J(\theta_1) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x)^{i} - y^{(i)}\right)^2$

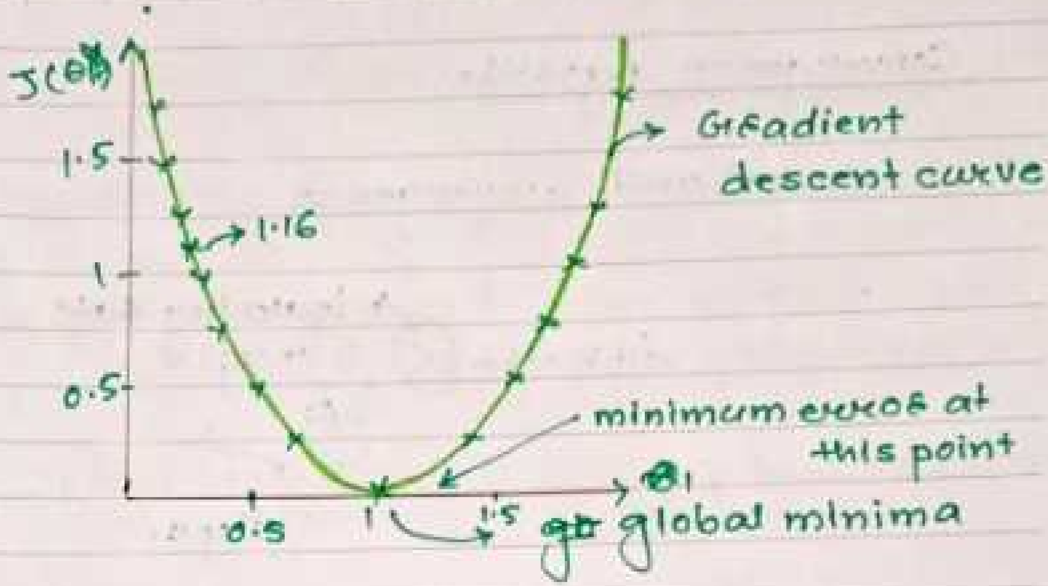
$= \frac{1}{3}\,[0+0+0]$

$= 0.$

when $\theta_1 = 0.5$

$J(\theta_1) = \frac{1}{3}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

$= \underline{1.16}.$

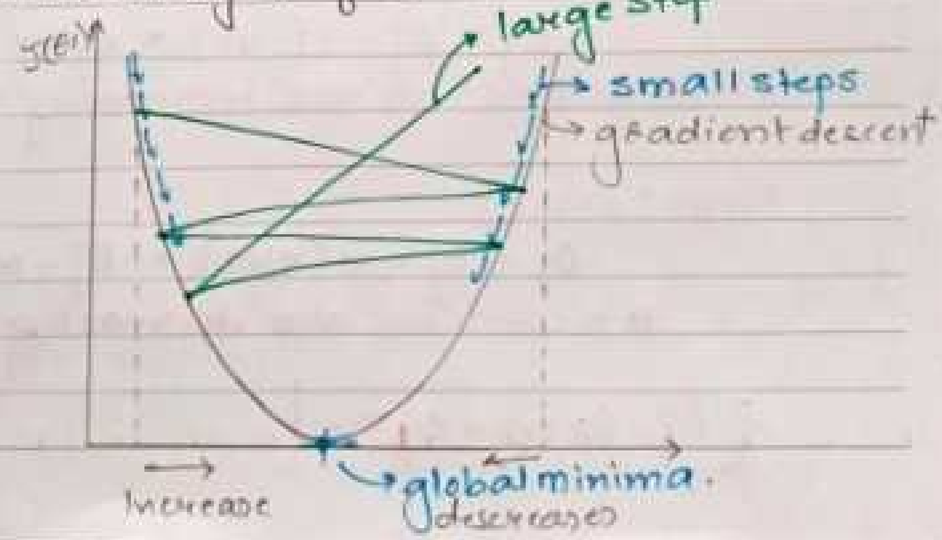


Objective: our main aim is to come near global minima.

we cannot change θ value manually, there should be some mechanism to change θ value to get the global minima.
To overcome this, we use Convergence algorithm.

Convergence algorithm:

- optimize the changes of θ1 value.



Convergence algorithm:

Repeat until convergence,

{

$$\theta_j := \theta_j - \alpha \underbrace{\frac{\partial}{\partial \theta_j} J(\theta_j)}_{\text{slope}}$$

α → learning rate

}

+ve or -ve slope:

right side of the line facing downwards
→ '-ve' slope

right side of the line facing upwards
→ '+ve' slope

'-ve' slope:

$$\theta_j = \theta_j - \alpha(-ve) = \theta_j + \alpha$$

It means we are increasing θ_j.

'+ve' slope:

$$\theta_j = \theta_j - \alpha(+ve) = \theta_j - \alpha$$

It means we are decreasing θ_j

$\boxed{\theta_j = \theta_0 \text{ and } \theta_1}$

α, Learning Rate

It decides the speed of the convergence.

If α is very small : It will take more time to reach global minima.

If α is very large : It will jump ~~here~~ and there, and won't reach to global minima.

α, should be around 0.001 for smaller steps.

convergence algorithm:

$$\underbrace{J(\theta_0,\theta_1)}_{\text{cost function}} = \frac{1}{m}\sum_{i=1}^{m}\Big(\underbrace{h_\theta(x)^{(i)}}_{\text{predicted}} - \underbrace{y^{(i)}}_{\text{actual or truth point}}\Big)^2$$

mean square error

m = no. of datapoint.

difference between cost function and loss function.

cost function: we find error for all the points and take average of it.

Loss function: we find error for observed points and if we find error for all the points then it is loss function.

$$\text{loss function} := \Big(\underbrace{h_\theta(x)^{(i)}}_{\text{predicted value}} - \underbrace{(y)^{i}}_{\text{actual value}}\Big)^2$$

$$= (\hat{y}_j - y_j)^2$$

* for every single point we need to find loss function.

To achieve global minima:

derivative w.r.t ~~to~~ θ_0, $j=0$

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_0} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2 \right\}$$

$$= \frac{\cancel{1}}{2m} \quad h_\theta(x) = \theta_0 + \theta_1 x$$

$$= \frac{\partial}{\partial \theta_0} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left[(\theta_0 + \theta_1 x)^i - y^i \right]^2 \right\}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x)^i - y^i \right) \times 1$$

$$\frac{\partial}{\partial x} \{(1+x) - y\}^2$$

$$= 2[(1+x)-y] \times 1$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x)^i - y^i \right)$$

derivative w.r.t. θ_1, $j=1$

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_1} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x)^{(i)} - y^{(i)} \right)^2 \right\}$$

$$= \frac{\partial}{\partial \theta_1} \left\{ \frac{1}{2m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x)^i - y^i \right)^2 \right\}$$

$$= \frac{1}{m} \sum_{i=1}^{m} \left((\theta_0 + \theta_1 x)^i - y^i \right) x.$$

Repeat untill convergence

{

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \cdot \sum_{i=1}^{m} \left(h_\theta(x)^i - y^i \right)^2$$

$$\theta_1 := \theta_1 - \frac{\alpha}{m} \sum_{i=1}^{m} \left(h_\theta(x)^i - y^i \right) x^{(i)}$$

}

* learning rate: speed of convergence

Types of cost function:

1. MSE: mean square error
2. MAE: mean absolute error
3. RMSE: root mean square error.

Mean Square Error

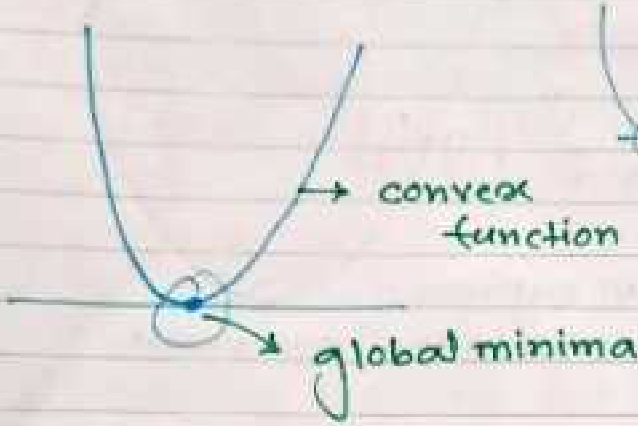
$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y - \hat{y})^2 \qquad \hat{y} = \theta_0 + \theta_1 x$$

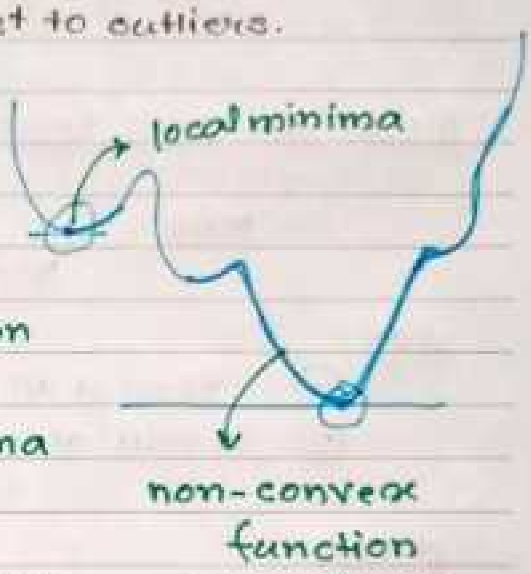
actual.

predicted.

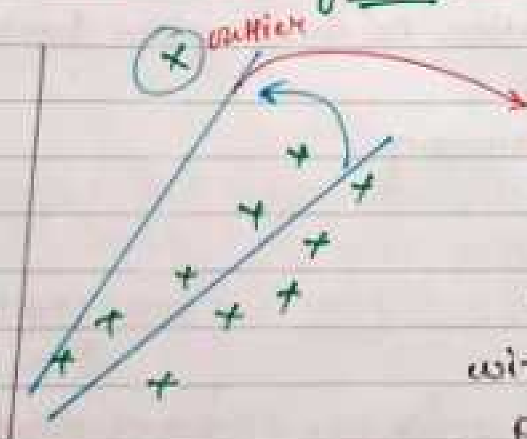
adv • This equation is differentiable.
• It also has one global minima.

disadv • This is robust to outliers.





∴ main adv. of <u>MSE</u> that it has convex function.



due to outlier, best fit line gets shifted.

Best fit line gets updated with huge margin in the presence of outlier.

Eg. (salary) → dependent feature
Experience → Independent feature

$(y-\hat{y})^2$ (lakhs)^2
↳ unit changing → time complexities increases.

we don't do scaling for dependent feature.

$(y-\hat{y})^2 \rightarrow$ squared $\rightarrow$ error $\rightarrow$ penalized & increased.

Mean Absolute Error

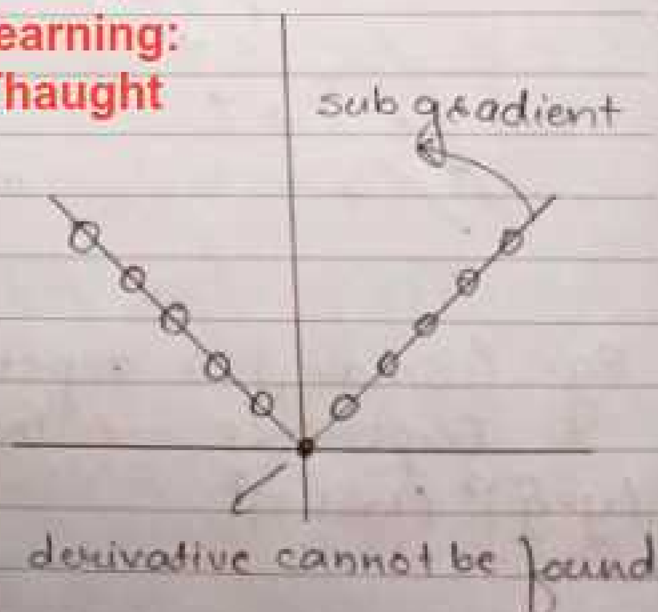
$$MAE = \frac{1}{m}\sum_{i=1}^{m} |y-\hat{y}|$$

adv:

1. Robust to outliers.
2. It will also be in the same unit.

disadv:

1. convergence usually takes more time. optimization is a complex task.
2. time consuming.



Root mean Square Error (RMSE)

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y-\hat{y})^2}$$

adv:

- It is differentiable.
- unit remain same.

disadv:

- Not robust to outliers.

Huber Loss Function

The Huber loss offers the best of both worlds by balancing the MSE and MAE together.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y-f(x))^2 & \text{for } |y-f(x)| \leq \delta, \\ \delta|y-f(x)| - \frac{1}{2}\delta^2 & \text{otherwise,} \end{cases}$$

It says: for loss values less than delta, use the MSE, for loss values greater than delta, use the MAE.

Using the MAE for larger loss values mitigates the weight that we put on outliers so that we still get a well-rounded model.

At the same time, we use the MSE for the smaller loss values to maintain a quadratic function near the centre.

This has the effect of magnifying the loss values as long as they are greater than 1.

once the loss for those data points dips below 1, the quadratic function down-weights them to focus the training on the higher-error data points.

Note: use the Huber loss any time you feel that you need a balance between giving outliers some weight, but not too much.

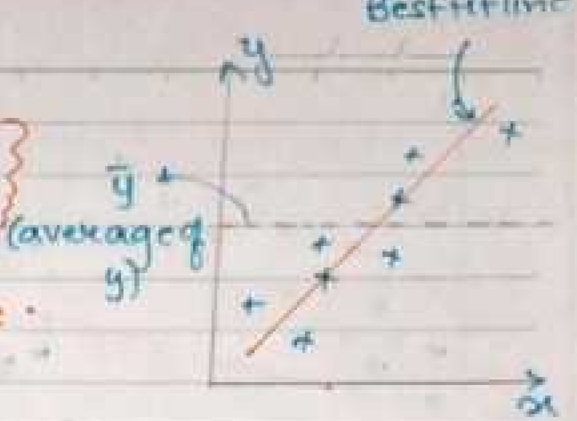
How can we check if a model is good or not?
→ using performance metrics.

Performance metrics

① R-squared
② Adjusted-R squared

R-squared : measures the performance of the model.

$$\boxed{\text{R-squared} = 1 - \frac{SS_{Res}}{SS_{Total}}}$$



SS_{Res} = Sum of Square Residuals

SS_{Total} = Sum of square average

$$\text{R-squared} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

low value

high value

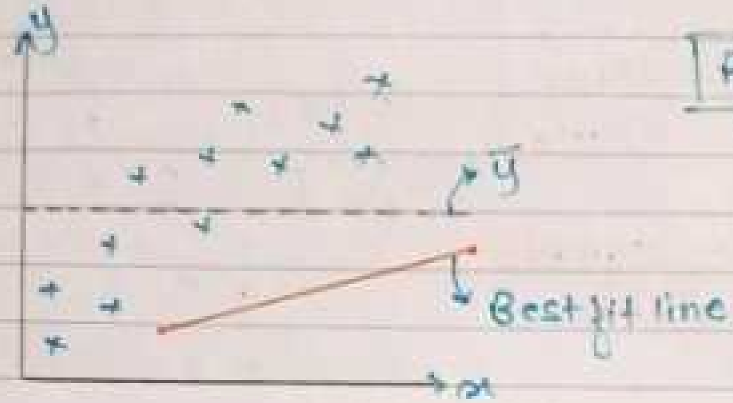
$\bar{y}$ = average of y.

$$= 1 - \left\{ \frac{\text{small value}}{\text{Bigger value}} \right\} \rightarrow \text{small value}$$

$\boxed{\text{R-squared} \leq 1}$

If R-squared = 0.85 = 85% accurate
= 0.75 = 75% accurate

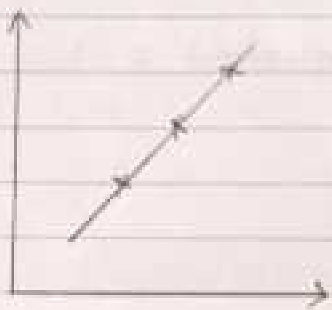
* If R-squared is '-ve' then the model is not working is good.



$\boxed{R^2 = -ve}$

Here $(y_i - \hat{y}) > (y_i - \bar{y}) \longrightarrow R^2$ will be negative.

$\boxed{R^2 = 1} \longrightarrow$



Adjusted R-squared

Eg.

No direct correlation

size of House | city location | No. of bedrooms | salary | Gender

Price

Before, when only size of house was present as an independent feature to get the price and has some R-squared value. But with addition of city location R^2 got increase. after that with addition of no. of bedroom, gender the value got increase, but there is ~~not~~ no direct correlation between gender and price, the value shouldn't increased.

R^2	features added	adj. R^2	Independent features
65%	size of house	63%	P=1
75%	location	73%	P=2
88%	no. of bedrooms	86%	P=3
90%	Gender	85%	P=4

no direct correlation w.r.t price

* very slightly increase, but this shouldn't happen.
To solve this problem we use Adjusted R^2.

$$\boxed{\text{Adjusted } R^2 = 1 - \frac{(1-R^2)(N-1)}{N-P-1}}$$

N = No. of datapoints
P = No. of independent features

* Adjusted R^2 is the best metrics to evaluate the model.

Overfitting and Underfitting

dataset
1000 datapoints → TRAINING (700) model training
1000 datapoints → TESTING (300) model test

overfitting and underfitting (Bias and Variance)

700

Training dataset

Train (500) → train the model

Validate (200) → hyperparameter tunning the model.

MODEL

Generalize Model

Train Data → very good accuracy (90%)

Test Data → very good accuracy (85%)

our aim is to get good test and train accuracy.

Train Data → very good accuracy 90% [low bias]

Test Data → very good accuracy 85% [low variance]

Overfitting

TRAIN ⟶ very good accuracy (90%) [low Bias]

TEST ⟶ Bad accuracy (50%) [High Variance]

Underfitting

TRAIN ⟶ model accuracy is low. [High bias]
TEST ⟶ model accuracy is low/high. [low or high variance]

* we can solve this issue by performing hyperparameter tunning or by increasing no. of dataset.

Regularization

Sometimes when we train a model it will start to overfit. A way to avoid overfitting data (especially for models like linear regressions that are heavily affected by outliers) we can use regularization. This will lead to a more general model that is technically less accurate but generalizes to the data better.

1. Ridge / L2 Regression

(used to reduce overfitting)

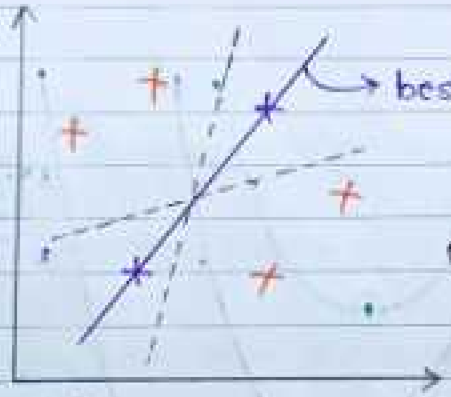
cost function = 0

Training data: low bias

Testing data: low/high variance

- If the data (test data) is near to best fit line then performance will be good. (low variance)
- If the test data is far (away) to best fit line then performance will be bad. (high variance)

aim: To reduce overfitting



→ we create multiple lines to improve performance of test data.

cost function:

$$\text{cost function} = \frac{1}{m}\sum_{i=1}^{m}\left\{h_\theta(x)^{(i)} - y^{(i)}\right\}^2 + \lambda(\text{slope})^2$$

λ = hyperparameter

Eg.

$h_\theta(x) = \theta_0 + \theta_1 x \longrightarrow$ slope $= \theta_1$

If multiple features are present, then

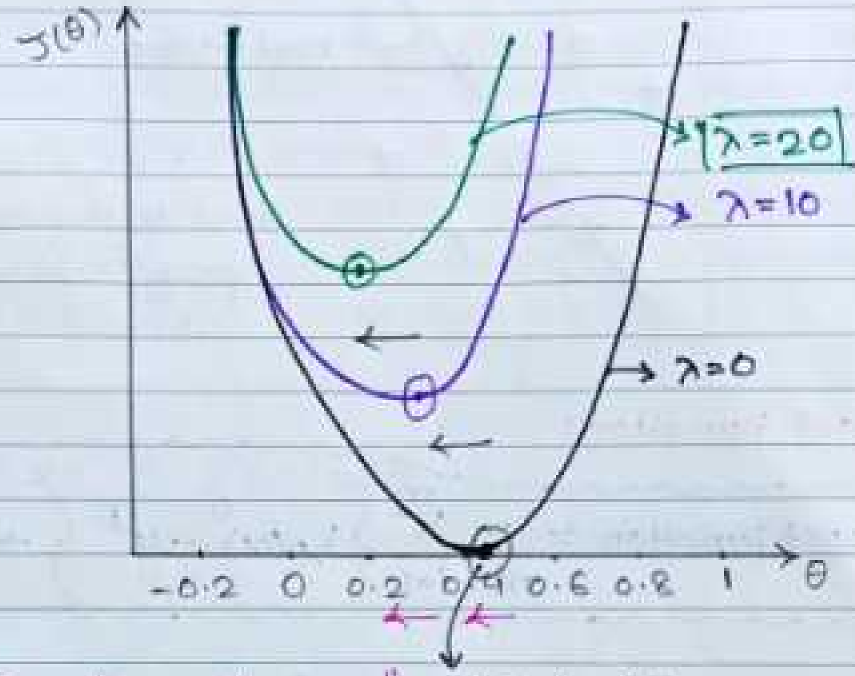
$$(\text{slope})^2 = \sum_{i=1}^{n}(\text{slope})^2$$

slope = slope of different lines.

If $\boxed{\lambda = 0}$,

cost function is same as linear regression's cost function.

Relationship between slope and λ



- Global minima gets shifted towards left with increase in λ.

$$\text{cost function} = 0 + (\text{slope})^2 \cdot \lambda$$

- change θ value to create another best fit line.

$$\boxed{\lambda \propto \frac{1}{\text{slope}}}$$

→ inversely proportional.

λ = It make sure that our line doesn't overfit.

$\lambda \geq 0$, is a complexity parameter that controls the amount of shrinkage:

the larger the value of λ, the greater the amount of shrinkage.
The coefficient are shrunk towards zero.

*θ value never becomes zero!

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + 0.95x_1 + 0.82x_2 + \underline{0.10x_3}$$

It will get deleted

∴ Ridge Regression is used to introduce bias to the data in order to generalize the data and increase bias.
This is useful if you don't have much training data.

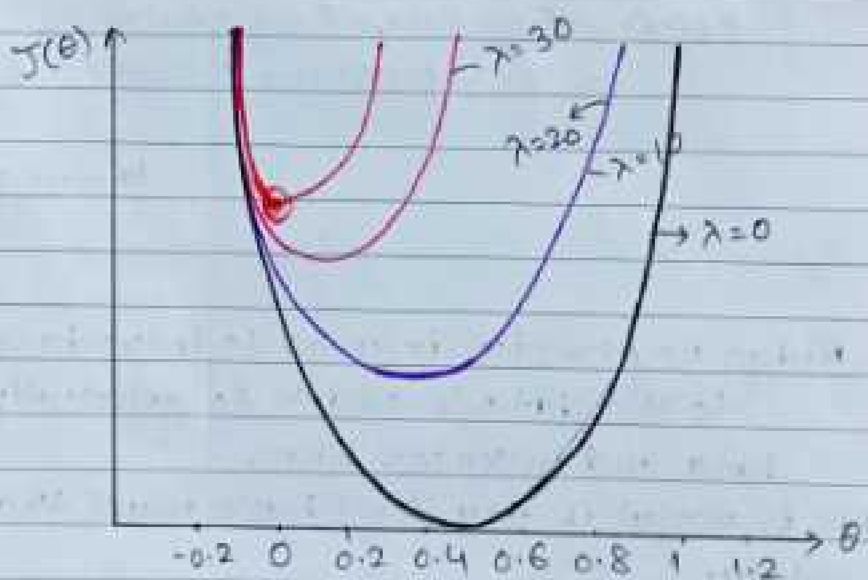
Lasso Regression

(L1 Regularization / L1 Norm)

- It is used to reduce the features. It helps in feature selection.

cost Function:

$$\boxed{\text{cost function} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x)^{(i)} - y^{(i)}\right)^2 + \lambda\sum_{i=1}^{m}|\text{slope}|}$$



$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

$\quad = \theta_0 + 0.54x_1 + 0.23x_2 + \underline{0.10x_3}$

least correlated.

- If data has outliers → use Ridge Regression.

Lasso = Least Absolute Shrinkage and Selection Operator Regression.

- Lasso regression tends to eliminate the weights of the least important features by setting their weights to zero.

Elastic Net

→ combination of L1 and L2 Regularization.

$$\boxed{\text{cost function} = \frac{1}{m}\sum_{i=1}^{m}\left\{h_\theta(x)^i - y^{(i)}\right\}^2 + \underbrace{\lambda(\text{slope})^2}_{L2} + \underbrace{\lambda|\text{slope}|}_{L1}}$$

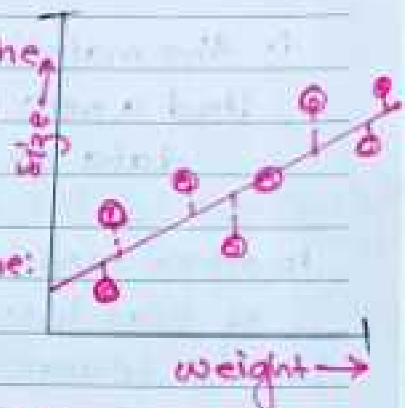
can be changed to MAE, RMSE, MSE.

Notes taken from John Starmer of Stat Quest

Youtube Videos

Regularization: Ridge (L2) Regression.

we find the line that results in the minimum sum of squared residuals.



∴ we end up with the eqn of the line:

$$\text{size} = 0.95 + 0.75 \times \text{weight}$$

y-intercept   slope

when we have of lot of measurements, we can be fairly confident that least squares line accurately reflects the relationship between size and weight.

But what if we only have two measurements?
we fit new line. since the new line overlaps the two data points, the minimum sum of squared residuals = 0.

∴ New line eqn: $\text{size} = 0.4 + 1.3 \times \text{weight}$

Sum of the squared residuals for testing data is large which means the new line has high variance.

In ML, new line (blue) is overfit to training data.

∴ the main idea behind Ridge Regression is to find a new line that doesn't fit the training data as well.

In other words, we introduce a small amount of Bias into how the new line is fit to the data but in return for that small amount of bias, we get a significant drop in variance.

∴ Ridge Regression can provide better long term predictions.

when least squares determines values for the parameters in this equation

↓

size = y-axis intercept + slope × weight

it minimizes....

the sum of the squared residuals.

In contrast,

when Ridge Regression determines values for the parameters in this equation....

↓

size = y-axis intercept + slope × weight

.... it minimizes

the sum of the squared residuals

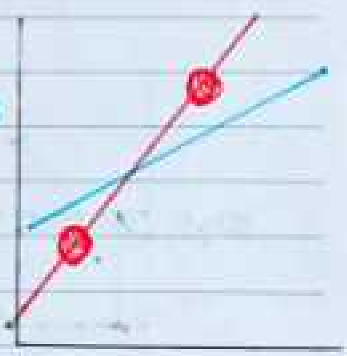
\+

λ × the slope²

λ → lambda.

this part adds a penalty to the traditional least square method.

and lambda(λ) determines how severe that penalty is.



→ the sum of squared residuals for the ~~residuals~~ least square fit is 0 (because the line overlaps the data points). and the slope is 1·3.

$$\therefore \quad 0 + \lambda \times (1.3)^2 = 0 + 1 \times (1.3)^2 = \underline{1.69} \qquad (\lambda = 1)$$

for blue line → slope = 0·8

$$(0.3)^2 + (0.1)^2 + \lambda \times (0.8)^2 \qquad \lambda = 1$$

alltogether 0·74

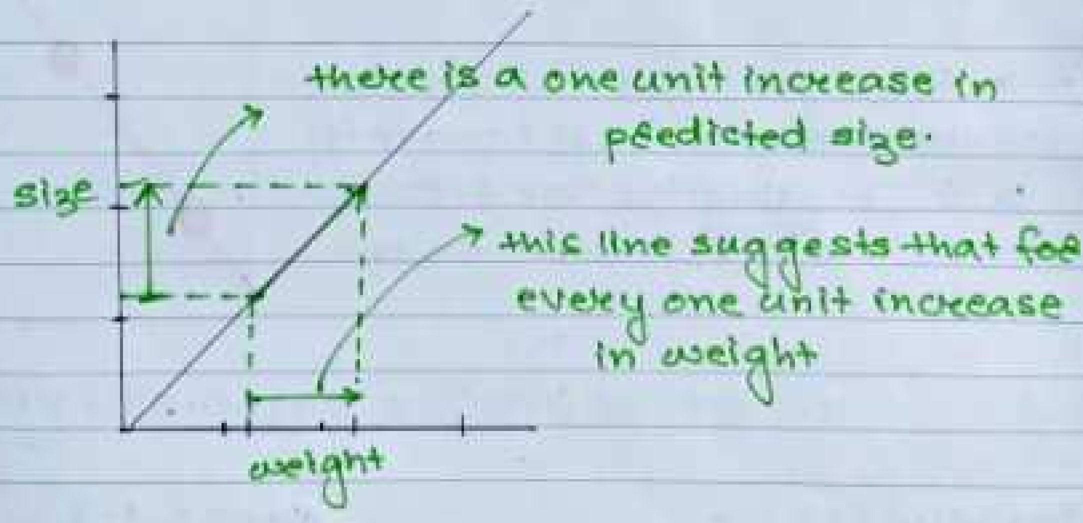
∴ Ridge Regression line :-

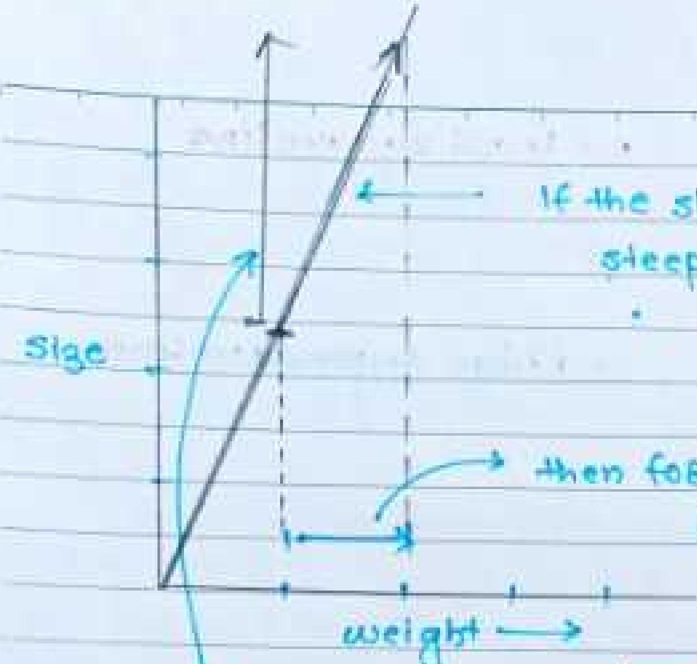
Red = 1·69 , Blue = 0·74

Thus, if we wanted to minimize the sum of the squared residuals plus the Ridge Regression penalty, we would choose the Ridge Regression Line over the least square line.

without the small amount of Bias that the penalty creates, the least squares fit has a large amount of Variance.

In contrast, the Ridge Regression line, which has the small amount of Bias due to the penalty, has less variance.





If the slope of the line is steeper

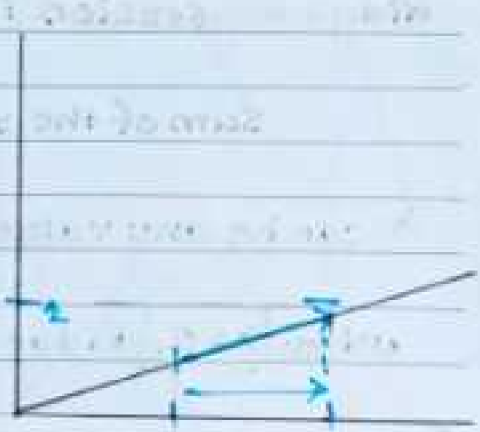
then for every one unit increase in weight

then the prediction size increases by over two units.

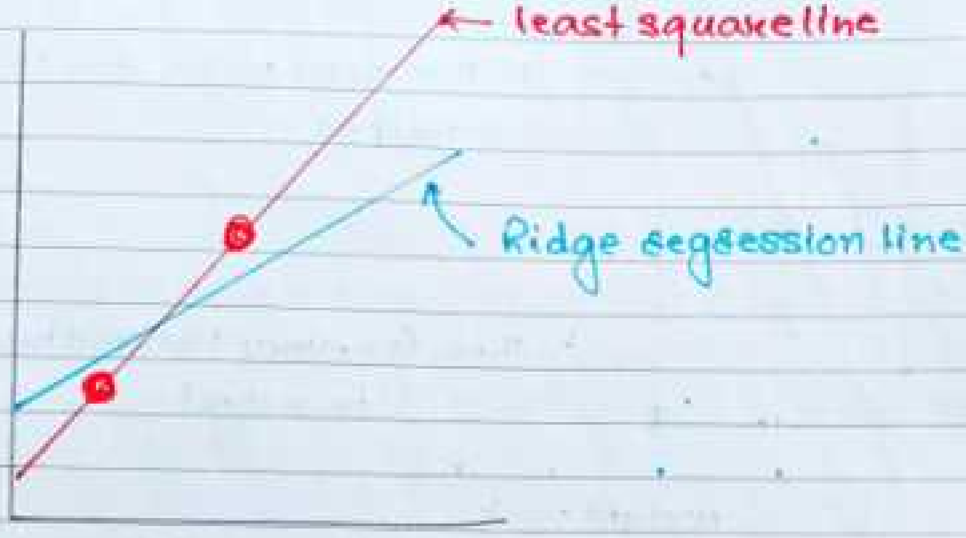
In other words, when the slope of the line is steep, then the prediction for size is very ~~sensitiving~~ sensitive to relatively small changes in weight.

when the slope is small, then for every one unit increase in weight

the prediction for size barely increases.



In other words, when the slope of the line is small, then predictions for size are much less sensitive to changes in weight.



The Ridge Regression penalty resulted in a line that has a smaller slope ...
which means that predictions made with the Ridge Regression line are less sensitive to weight than the least square line.

Ridge Regression (RR)

Sum of the squared residuals + $\lambda \times (\text{slope})^2$

λ can be any value from 0 to positive infinity.

when $\boxed{\lambda = 0,}$ Ridge regression line = least square line

$\boxed{\lambda = 1}$ RR ended up with a smaller slope than the least square line.

and the larger we make λ, the slope gets asymptotitically close to 0.

So, the larger λ gets, our predictions for size becomes less and less sensitive to weight.

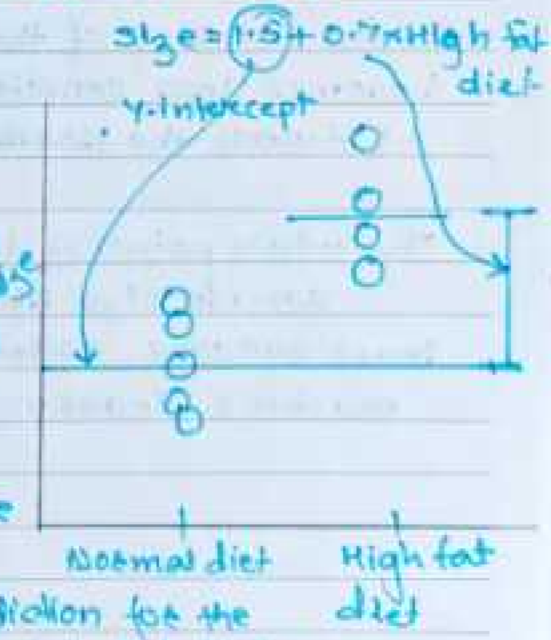
So how do we decide what value to give λ?

→ we just try a bunch of values for λ and use cross-validation, typically 10-fold cross validation, to determine which one results in the lowest variance.

.....uptill now RR was for continuous variable.

However, RR also works when we use discrete variable.

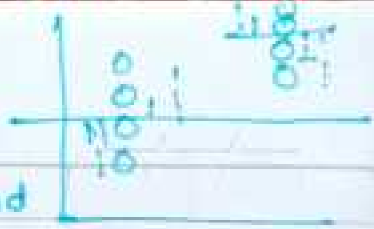
Discrete variable:



y-intercept → corresponds to the average size of the mice on the Normal diet.

size = 1.5 + 0.7 x High fat diease

sum of these two is the prediction for the size of the mice on the High fat diet.



these distance between the data and the means are minimized. .

when RR determines value for the parameters in the equation...

... it minimizes →

the sum of the squared residuals

+

$\lambda \times (\text{diet difference})^2$

$\lambda=0$, least squared-error = RR line

$\lambda=1$, then only way to minimize the whole eqn is to shrink diet distance down.

In other words, as λ gets larger; our prediction for the size of the mice on the high fat diet becomes less sensitive to the difference between the Normal diet and High-fat diet.

The whole point of doing RR is because small sample size like these can lead to poor least squares estimates that result in terrible machine learning predictions.

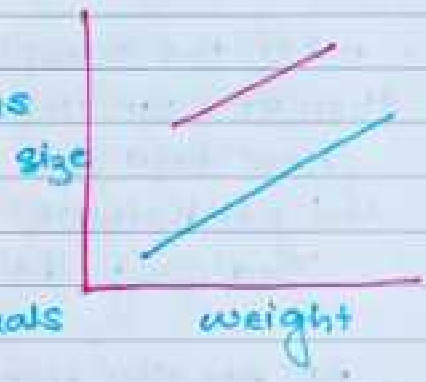
Ridge Regression can also be applied to Logistic Regression.

= the sum of the likelihoods + $\lambda(\text{slope})^2$

Note:- when applied to Logistic Regression, Ridge Regression optimizes the sum of the likelihoods instead of the squared residuals because Logistic Regression is solved using maximum likelihood.

" Ridge Regression helps reduce variance by shrinking parameters and making our predictions less sensitive to them

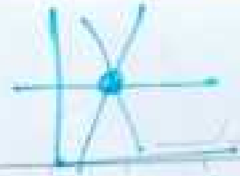
In general, RR penalty contains all of the parameters except for the y-intercept.



the sum of the squared residuals

+

$\lambda(\text{slop}^2 + \text{diet distance}^2)$.



least squares can't find a single optimal solution, since any line line that goes through the dot will minimize the sum of the squared residuals.

but RR can find a solution with cross validation and the RR ~~policy~~ penalty that favours smaller parameter values.

↓

sum of the squared residuals

+

$\lambda *(\text{slope})^2$

Summary:

- when the sample sizes are relatively small, then RR can improve predictions made from new data (i.e. reduce variance) by making the predictions less sensitive to the Training data.

RR penalty itself is λ times the sum of all squared parameters, except for the y-intercept. and λ is determined using cross validation.

Lasso Regression: (L1)

Ridge Regression Penalty = $\lambda \times (\text{slope})^2$

* Lasso Regression →

sum of all the squared residuals + $\lambda \times |\text{slope}|$

Lasso Regression Penalty contains all of the estimated parameters except for the y-intercept.

∵ Ridge and Lasso regression shrink parameters, they don't have to shrink them all equally.

Big difference between Ridge and Lasso Regression is that Ridge Regression can only shrink the slope asymptotically close to 0 while Lasso regression can shrink the slope all the way to 0.

LR can exclude useless variables from equations, better than RR at reducing the variance in models that contain a lot of useless variables.

Elastic Net Regression:

Elastic-Net Regression starts with least-squares then combines the Lasso Regression penalty with the Ridge Regression penalty.

$$\text{sum of the squared residuals}$$
$$+$$
$$\lambda_1 \times |\text{variable}_1| + \ldots\ldots + |\text{variable}_n|$$
$$+$$
$$\lambda_2 \times (\text{variable}_1)^2 + \ldots\ldots + (\text{variable}_n)^2$$

Note: LR and RR penalty get their own λs.

The hybrid Elastic Net Regression is especially good at dealing with situations when there are correlations between parameters.

This is because on it's own, Lasso Regression tends to pick just one of the correlated terms and eliminates the others whereas RR tends to shrink all of the parameters for the correlated variables together.

By combining LR and RR,
Elastic-Net Regression groups and shrinks the parameters associated with the correlated variables and leaves them in eqn or removes them all at once.

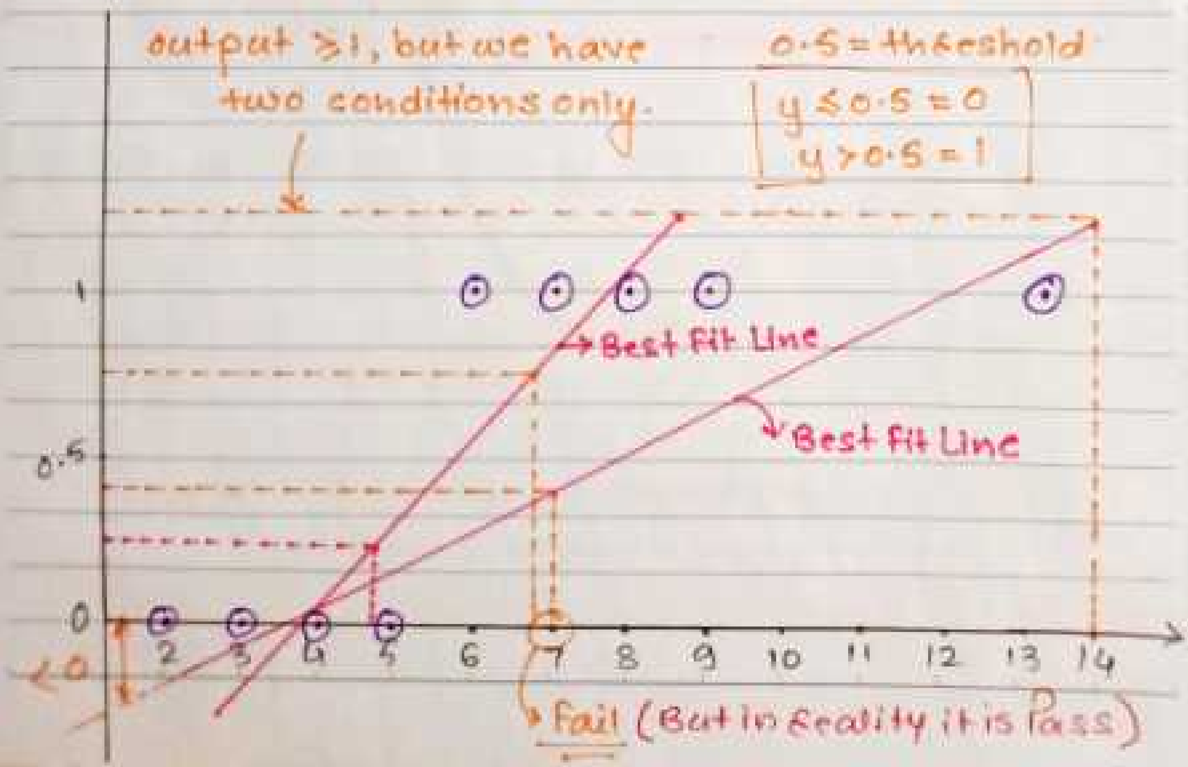
Logistic Regression

- classification problem

Eg.
dataset:

Study hours	Play hours	O/P (Pass/Fail)
2	8	Fail
3	7	Fail
6	3	Pass
outliers → 1	4	Pass

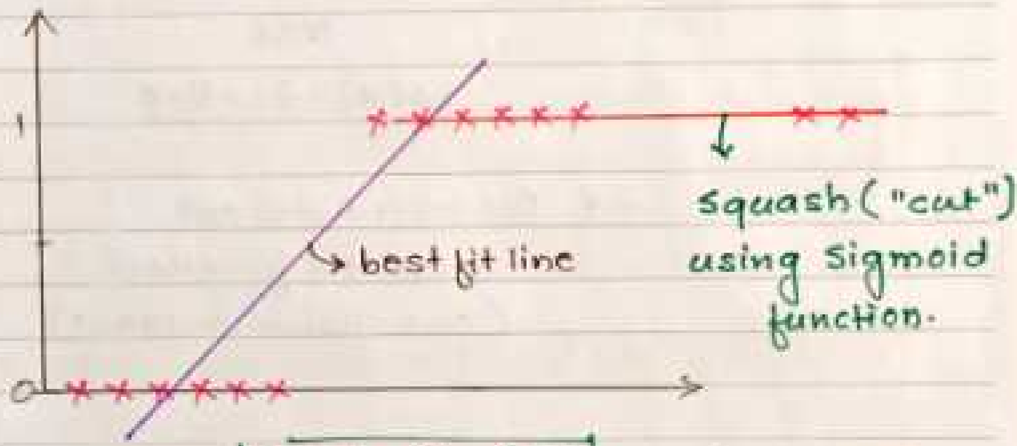
* we cannot perform regression, we need to perform classification into Pass / Fail.



why we use logistic Regression when we can solve classification problem using Linear Regression?

→ • due to outliers best fit line gets change and results will be wrong.

- we cannot remove outliers always.
- Threshold can't be changed, once fixed.
- In logistic we squash ("cut") the line. we will not change the line.



$\boxed{h_\theta(x) = \theta_0 + \theta_1 x}$

⇓

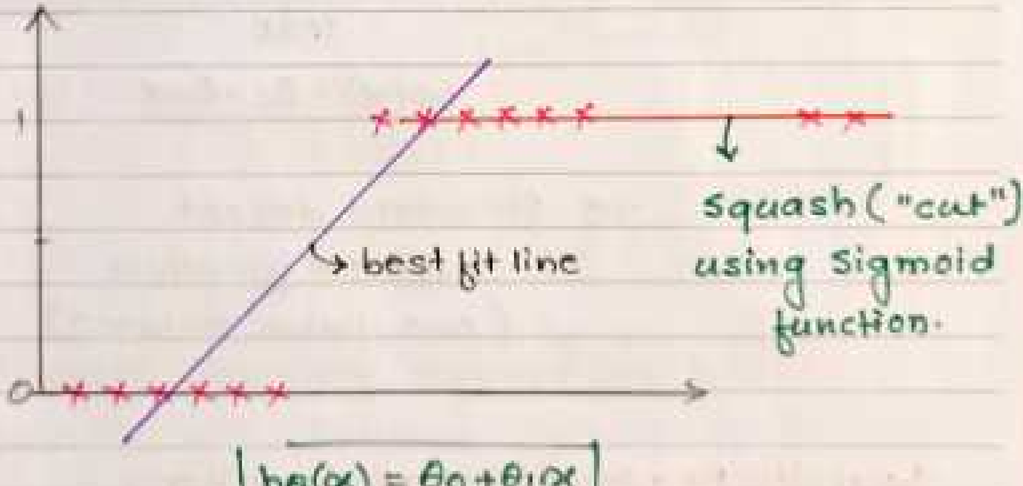
sigmoid function → output will range betn 0 and 1.

so, even if a person study for 7hrs
y<0.5
∴ the model will show Fail.
∵ we cannot used Linear Regression in this type of problem statement.

n we can
sing

change and

- we cannot remove outliers always.
- Threshold can't be changed, once fixed.
- In logistic we squash ("cut") the line. we will not change the line.



$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x}$$

⇓

Sigmoid function → output will range betn 0 and 1.

Sigmoid Function:

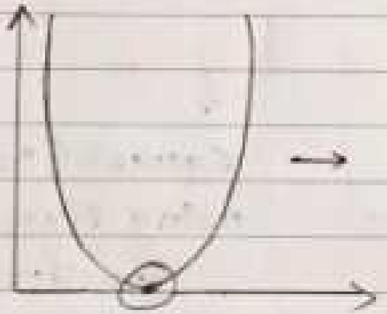
$$\boxed{\text{sigmoid function} = \frac{1}{1+e^{-z}}} \rightarrow \text{bet}^{n}\ 0\ \&\ 1.$$

1. create a best fit line.
2. squashing → sigmoid function

Linear Regression cost function:

$$J(\theta_0,\theta_1) = \frac{1}{2m} \underbrace{\sum_{i=1}^{m} \left\{ h_\theta(x)^{(i)} - y^{(i)} \right\}^2}_{MSE}$$

$\because h_\theta(x) = \theta_0 + \theta_1 x$



→ Gradient desent
convex function
(one global minima)

Logistic Regression cost function:

steps:
① create best fit line
② apply squashing using sigmoid function.

$$J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left\{ h_\theta(x)^{(i)} - y^{(i)} \right\}^2$$

sigmoid function

$$h_\theta(x) = \sigma(\theta_0 + \theta_1 x)$$

~~let $z = h_\theta(x) = \theta_0 + \theta_1 x$~~

$$\text{let } z = \theta_0 + \theta_1 x$$

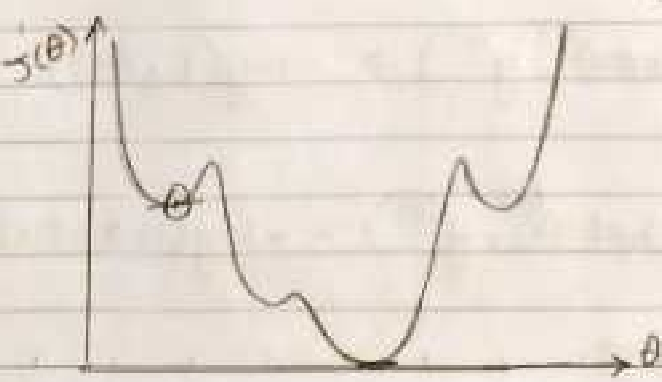
$$h_\theta(x) = \sigma(z)$$

$$h_\theta(x) = \frac{1}{1 + e^{-z}}$$

$$\therefore \quad h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}$$

$$\therefore \quad \boxed{h_\theta(x) = \frac{1}{1 + e^{-(\theta_0 + \theta_1 x)}}}$$

but, after applying sigmoid function, cost function will become non-convex function and have a chances to get local minima.



← Non-convex function.

- change the cost function to solve the convexity problem.

- Log Loss Cost Function

$$\text{cost function} = \begin{cases} -\log(h_\theta(x)), & \text{if } y=1 \\ -\log(1-h_\theta(x)), & \text{if } y=0 \end{cases}$$

→ convex function

$$h_\theta(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x)}}$$

$$\text{cost}(h_\theta(x)^{(i)}, y^{(i)}) = -y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$$

→ This will never give local minima.

if $y=1$,

$$\text{cost}(h_\theta(x)^{(i)}, y^{(i)}) = -\log(h_\theta(x))$$

if $y=0$,

$$\text{cost}(h_\theta(x)^{(i)}, y^{(i)}) = -\log(1-h_\theta(x))$$

y = truth value

minimize cost function $J(\theta_0, \theta_1)$ by changing θ_0, θ_1.

Convergence Algorithm:

Repeat Convergence

$$\left\{ \theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \right\}$$

for $j=0$ and $j=1$

- by default take threshold = 0.5
 using ROC and TOC curve, we can define threshold.

Performance Metrics

1. Confusion matrix
2. Accuracy
3. Precision
4. Recall
5. F-Beta Score

dataset.

f_1	f_2	O/P	P= $\hat{y}$ (model predictn)
–	–	0	1
–	–	1	1
–	–	0	0
–	–	1	1
–	–	1	1
–	–	0	1
–	–	1	0

Confusion matrix

← Actual value (y)

1 → correct prediction
0 → wrong prediction

	1	0
1	3	2
0	1	1

↑ predicted value ($\hat{y}$)

← Actual value (y)

Predicted value ($\hat{y}$) ↑	1	0
1	TP	FP
0	FN	TN

correct match { TP: True Positive
TN: True Negative

wrong match { FP: False Positive
FN: False Negative

Accuracy

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

Eg. $\text{accuracy} = \frac{3+1}{3+2+1+1} = \frac{4}{7} = 57\%$

dataset → Binary classification

↓

1000 datapoints

↙ ↘

900 (1) 100 (0) ← Imbalanced dataset

dumb model → 1 ⇒ we get 90% accuracy.

- If the accuracy is 90%, it is not sufficient. model is not good. To overcome this problem we can use Recall and Precision.

· Precision:

$$\boxed{\text{Precision} = \frac{TP}{TP + FP}}$$

Predicted ↓ / Actual →	1	0
1	TP	FP
0	FN	TN

out of all the actual values how many are correctly predicted.

· our aim is to reduce False Positive (FP)

Eg. ① mail → spam or harm

TP: actual → spam and predicted → spam
(Good)

FN actual → spam and predicted → harm
(not good)

FP: actual → harm and ~~actual~~ predicted → spam
(mail is not spam)
(critical problem)
focus on reducing FP.

② Diabetes or Not diabetes

Actual → diabetes and Predicted → Not diabetes
critical problem → reduce FN.

Recall :

$$\boxed{Recall = \frac{TP}{TP+FN}}$$

→ out of all the predicted values how many are currently predicted.

Eg. Tommorow the stock market is going to crash.

Two points
→ consumers → FN ↓
→ companies → FP ↓↓ (can take certain decision)

1 ← Actual 0

predicted ↑	1	0
1	TP	FP
0	FN	TN

for FP companies can take action.
company (action: sale shares and discounted price)

consumer

F-Beta score

$$\boxed{F\text{-}Beta\ score = \frac{(1+\beta^2)\,Precision \times Recall}{\beta^2 \cdot Precision + Recall}}$$

① If FP and FN are both important

$\beta = 1$

$\therefore$ $$\boxed{F1 \text{ score} = \frac{2 \cdot P \times R}{P + R}}$$

② If FP is more important than FN

$\beta = 0.5$

$$\boxed{F\text{-}0.5 \text{ score} = \frac{(1+0.25) P \times R}{0.25 \times P + R}}$$

③ If FN >> FP, (FP is less important than FN),

$\beta = 2$

$$\boxed{F2 \text{ score} = \frac{(1+4) P \times R}{4 \times P + R}}$$

In-depth Insights

Logistic Regression

Classification

Classification problem, is just like the regression model problem, except that the values we now want to predict take on only a small number of discrete values.

$y \in \{0,1\}$ 0 : "-ve class"
Binary class 1 : "+ve class"

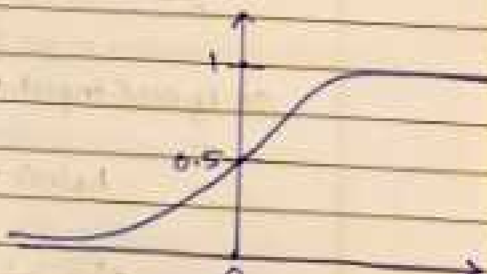
Hypothesis Representation

Intuitively, it doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that $y \in \{0,1\}$. To fix this, let's change the form for our hypothesis $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$.

This accomplished by plugging $\theta^T x$ into the logistic function.

Our new form uses the "Sigmoid Function" also called the "Logistic Function".

$$h_\theta(x) = g(\theta^T x)$$
$$z = \theta^T x$$
$$g(z) = \frac{1}{1+e^{-z}}$$



Function $g(z)$, shown here, maps any real number to the (0,1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

$h_\theta(x)$ will give us the probability that our output is 1.

Eg. if $h_\theta(x) = 0.7 \longrightarrow$ output : 1
thus $h_\theta(x) = 0.3 \longrightarrow$ 0

papergrid

Date: / /

$h_\theta(x) = P(y=1|x;\theta) = 1 - P(y=0|x;\theta)$

$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$

Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows.

$h_\theta(x) \geq 0.5 \rightarrow y=1$

$h_\theta(x) < 0.5 \rightarrow y=0$

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

$g(z) \geq 0.5$ when $z \geq 0$

Remember:

$z=0,\ e^{0}=1 \Rightarrow g(x)=1/2$

$z \rightarrow \infty,\ e^{-\infty} \rightarrow 0 \Rightarrow g(x)=1$

$z \rightarrow -\infty,\ e^{\infty} \rightarrow \infty \Rightarrow g(x)=0$

So if our input to g is $\theta^T x$, then that means

$h_\theta(x) = g(\theta^T x) \geq 0.5$ when $\theta^T x \geq 0$

$\therefore$ $\theta^T x \geq 0 \rightarrow y=1$

$\theta^T x < 0 \rightarrow y=0$

The decision boundary is the line that seperates the area where $y=0$ & $y=1$. It is created by our hypothesis function.

Eg. $\theta = \begin{bmatrix} 5 \\ -1 \\ 0 \end{bmatrix}$

$$y = 1 \text{ if } 5 + (-1)x_1 + 0 \cdot x_2 \geq 0$$
$$5 - x_1 \geq 0$$
$$-x_1 \geq -5$$
$$x_1 \leq 5$$

In this case, our decision boundary is a straight vertical line placed on the graph where $x_1 = 5$, and everything to the left of that denotes $y=1$, while everything to the right denotes $y=0$.

again, the input to the sigmoid function $g(z)$ (e.g. $\theta^T X$ doesn't need to be linear, could be a function that describes a circle (e.g. $z = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$) or any shape to fit our data.

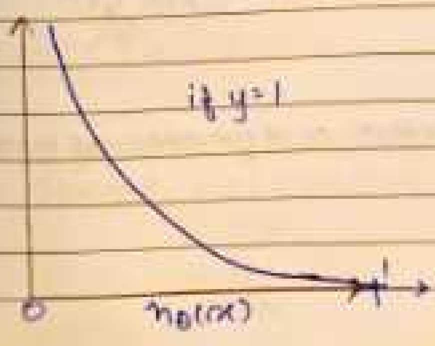
Cost Function

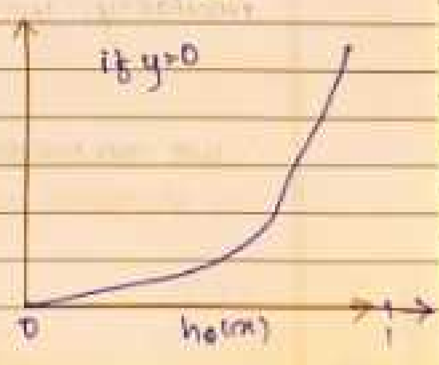
we cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima.
In other words, it will not be a convex function.

Cost function for logistic function:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \text{cost}(h_\theta(x^i), y^i)$$

$$\text{cost}(h_\theta(x^i), y) = -\log(h_\theta(x)) \quad \text{if } y=1$$
$$\text{cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \quad \text{if } y=0$$





papergrid

Date: / /

$cost(h_\theta(x), y) = 0$ if $h_\theta(x) = y$

$cost(h_\theta(x), y) \rightarrow \infty$, if $y=0$ and $h_\theta(x) \rightarrow 1$

$cost(h_\theta(x), y) \rightarrow \infty$ if $y=1$ and $h_\theta(x) \rightarrow 0$

If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.

If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.

Note that writing the cost function in this way guarantee that $J(\theta)$ is convex for logistic ~~function~~ regression.

Simplified Cost function and Gradient Descent

We can compress our cost function's two conditional cases into one case.

$$Cost(h_\theta(x), y) = -y \cdot \log(h_\theta(x)) - (1-y) \cdot \log(1-h_\theta(x))$$

Thus, when we substitute $y=1$ in abv eqn,

$cost(h_\theta(x), y) = -y \cdot \log(h_\theta(x))$

similarly, we get another term when $y=0$.

$\therefore$ we can write our entire cost function as follows:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y^{(i)} \cdot \log(h_\theta(x^i)) + (1-y^i) \cdot \log(1-h_\theta(x^i)) \right]$$

A vectorized Implementation is:

$$h = g(X\theta)$$

$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1-y)^T \cdot \log(1-h) \right)$$

Gradient Descent:

General form of gradient descent is:

Repeat {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

}

we can workout the derivative part using calculus to get:

Repeat {

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) \cdot x_j^{(i)}$$

}

Notice that this algorithm is identical to the one we used in linear regression. we still have to simultaneously update all values in theta. ($h_\theta(x)$ is different)

A vectorized implementation is:

$$\theta := \theta - \frac{\alpha}{m} X^T (g(X\theta) - \vec{y})$$

Advanced Optimization

cost function $J(\theta)$. want $\min_\theta J(\theta)$.

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ for $j = 0, 1, \ldots, n$

Date: / /

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS.

advantages :- ① No need to manually pick α.
② often faster than gradient descent

disadv^n :- more complex to implement.

Multiclass classifications

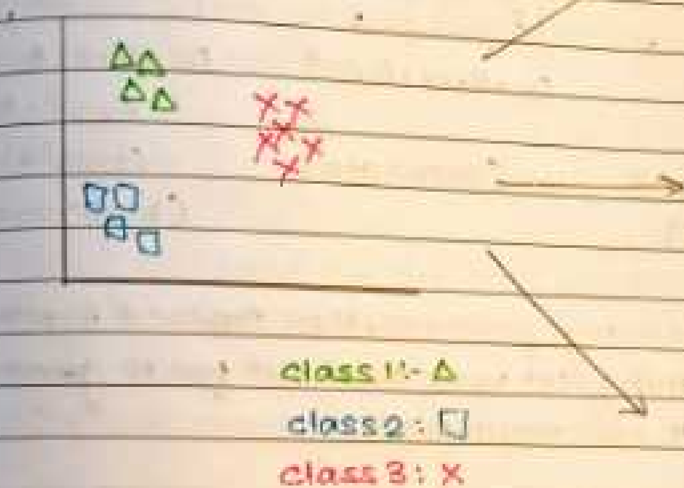
Now we will approach the classification of data when we have more than two categories.
instead of $y = [0,1]$ we will expand our definition so that $y = [0,1,\ldots,n]$

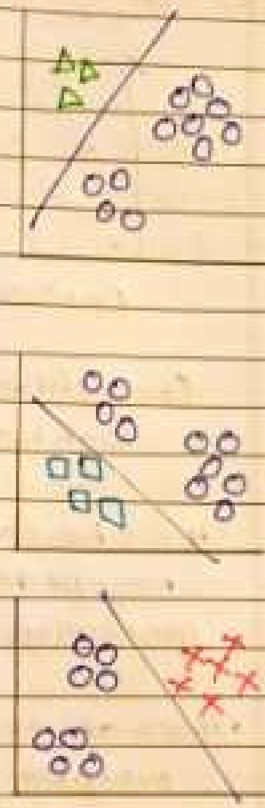
since $y = [0,1,\ldots,n]$, we divide our problem into n+1 (+1 bcz the index starts at 0) binary classification problems; In each one, we predict the probability that 'y' is a ~~member~~ member of one of our classes.

$$y \in \{0,1,\ldots,n\}$$
$$h_\theta^{0}(x) = P(y=0|x;\theta)$$
$$h_\theta^{1}(x) = P(y=1|x;\theta)$$
$$\vdots$$
$$h_\theta^{n}(x) = P(y=n|x;\theta)$$
$$\text{prediction} = \max_i (h_\theta^{i}(x))$$

we are basically choosing one class and then lumping all the others into a single second class. we do this repeatedly, applying binary logistic regression to each case, and thus use the hypothesis that returned the highest value as our prediction.

one-vs-all (one-vs-real)





$h_\theta^{i}(x) = P(y=i|x;\theta)$ $(i=1,2,3)$

To summarize :

Train a logistic regression classifier $h_\theta(x)$ for each classifier i to predict the probality that $y=i$

on a new input x, to make a prediction, pick the class that maximizes

$$\max_i h_\theta^{(i)}(x)$$

Regularization

$\theta_0+\theta_1 x$ (underfitting or high bias)

$\theta_0+\theta_1 x+\theta_2 x^2$ (accurate)

$\theta_0+\theta_1 x+\theta_2 x^2+\theta_3 x^3+\theta_4 x^4$ (overfit) (high variance)

If we have too many features, then learned hypothesis may fit training set very well but fail to ~~generate~~ generalize to new examples.

There are two main options to address the issue of overfitting :

1. Reduce the numbers of features:
   - manually select which features to keep.
   - use a model selection algorithm

2. Regularization
   - keep all the features, but reduce the magnitude of parameters θ_j.
   - Regularization works well when we have a lot of slightly useful ~~resource~~ features.

Cost functions

If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our

papergrid

Date: / /

function carry by increasing their cost.

Eg.

We wanted to make the following function more quadratic:

$$\theta_0 + \theta_1 x + \theta_2 x^2 + \underbrace{\theta_3 x^3 + \theta_4 x^4}_{\text{eliminate}}$$

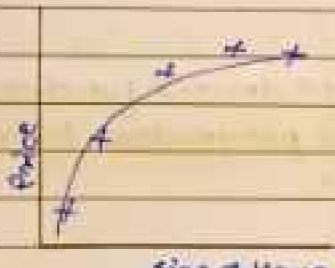
~~we will~~

without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our cost function.

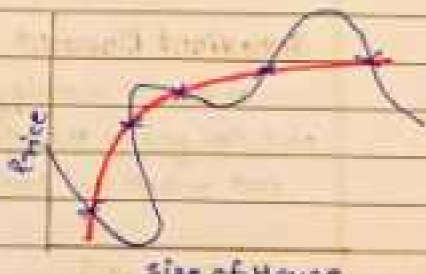
$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2$$

Now, in order for the cost function to get close to zero, we will have to reduce the values of θ_3 and θ_4 to near zero. This will in turn greatly reduce the values of $\theta_3 x^4$, $\theta_4 x^4$ in our hypothesis function.

As a result, we see that new hypothesis (Pink Graph) looks like a quadratic function but fits the data better due to extra small items $\theta_3 x^3$ and $\theta_4 x^4$.



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



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3, θ_4 really small.

$$\min_\theta \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i\right)^2 + \underline{1000\,\theta_3^2} + \underline{1000\,\theta_4^2}$$

pink graph shows this complete equation.

we could also regularize all of our theta parameters in a single summations as:

$$\min_{\theta} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right)^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

The λ, or lambda, is the Regularization parameter. It determines how much the costs of our theta parameters are inflated.

using the above cost function with the extra summation, we can smooth the output of our hypothesis function to reduce overfitting.

If lambda is choosen to be too large, it may smooth out the function too much and cause underfitting.

Regularized Linear Regression

Gradient Descent

we will modify gradient descent function to separate out θ_0 from the rest of the parameters because we do not want to penalize θ_0.

Repeat {

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right) \cdot x_0^i$$

$$\theta_j := \theta_j - \alpha \left[\left(\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^i) - y^i \right) x_j^i \right) + \frac{\lambda}{m} \theta_j \right]$$

$j \in \{1, 2, \ldots, n\}$

The term $\frac{\lambda}{m}\theta_j$ performs our regularization. with some manipulation our update rule can also be represented as:

$$\theta_j := \theta_j\left(1-\alpha\frac{\lambda}{m}\right) - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)\cdot x_j^i$$

↳ always will be less than 1

Intuitively we can see it as reducing the value of θ_j by some amount on every update.
so 2nd term is exactly same as it was before.

Normal Equation:-

To add in regularization, the equation is the same as our original, except that we add another term inside the parantheses:

$$\theta = \left(X^TX + \lambda\cdot L\right)^{-1}X^Ty$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

→ always zero

L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else.

It should have dimension $(n+1)\times(n+1)$. Intuitively, this is the ~~identity~~ identity matrix (though we are not including x_0), multiplied with a single real no. λ.

If $m<n$, then X^TX is non-invertible. However, when we add the term $\lambda\cdot L$, then $X^TX+\lambda\cdot L$ becomes invertible.

Regularized Logistic Regression

cost function:

cost function for logistic regression was:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[y^i.\log(h_\theta(x^i)) + (1-y^i).\log(1-h_\theta(x^i))\right]$$

we can regularize this equation by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i.\log(h_\theta(x^i)) + (1-y^i).\log(1-h_\theta(x^i))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

means to explicitly exclude the bias term θ_0. i.e. the θ vector is indexed from 0 to n (holding n+1 values, θ_0 through θ_n).
and this sum explicitly skips θ_0, by running from 1 to n, skipping 0.
Thus, when computing the equation, we should continuously update the two following equations.

Gradient Descent

repeat {

$$\theta_0 := \theta_0 - \alpha.\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i).x_0^i$$

$$\theta_j := \theta_j - \alpha\underbrace{\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)x_j^i + \frac{\lambda}{m}\theta_j\right]}_{\frac{\partial}{\partial\theta_j}J(\theta)} \qquad j=(1,2,\dots,n)$$

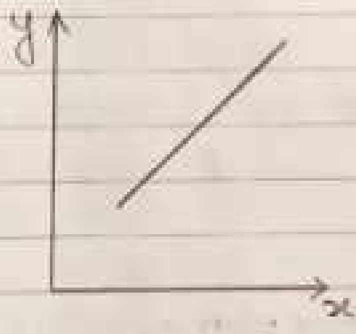
Support Vector Machine
(SVM)

It can solve both classification and Regression problem.

1. classification $\longrightarrow$ SVC (Support Vector classifier)

2. Regression $\longrightarrow$ SVR (Support Vector Regressor)

some basics:



Equation of line:

$y = mx + c$ OR

$y = \beta_0 + \beta_1 x$ OR

$ax + by + c = 0$

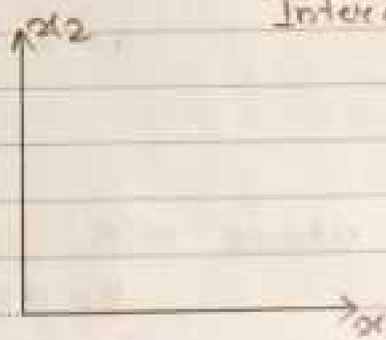
$$\therefore \; y = \underbrace{\frac{-a}{b}}_{\text{coefficient}} x - \underbrace{\frac{c}{b}}_{\text{Intercept}}$$

$ax_1 + bx_2 + c = 0$

$w_1 x_1 + w_2 x_2 + b = 0$

$\therefore \; w^T x + b = 0$

If line passes through origin $\therefore \; w^T x = 0$

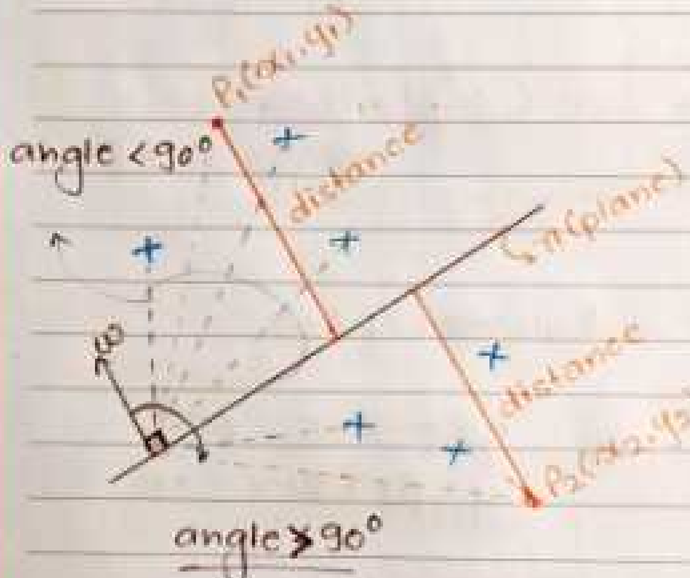


matrix multiplication: $\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \begin{bmatrix} x_1 & x_2 \end{bmatrix}$

$w^T x + b = 0$. ($w^T x$: w Transpose x)

Equation of line passing through origin is:

$\boxed{w^T x = 0}$



we have to find the distance of point from the plane.

(π = line in 2D and plane in 3D)

Distance of a point to the plane,

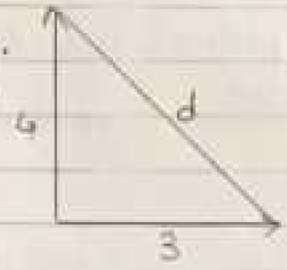
$$\boxed{\text{distance}(d) = \frac{w^T P_1}{\|w\|}}$$

$= \|w\| \cdot \|P\| \cdot \cos\theta$

where $w^T P_1$: w Transpose P_1, w : vector.
$\|w\|$: magnitude of w

unit vector: A vector which has a magnitude of 1 is basically called unit vector.

Eg.



Now,

$d = \sqrt{3^2 + 4^2}$

$= \sqrt{25}$

$\therefore$ $\underline{d = 5}$

where vector, $\hat{d} = \frac{d}{\|d\|}$ → magnitude

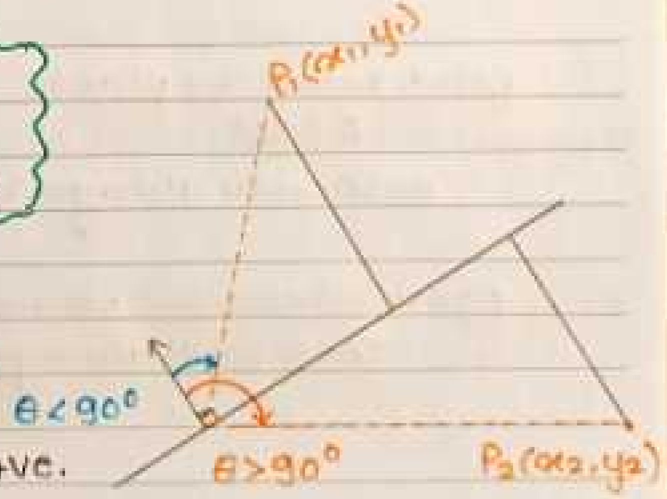
$\left(3/5, 4/5\right) = d = \sqrt{(3/5)^2 + (4/5)^2} = \sqrt{25/25}$

$= 1.$

$\therefore$ unit vector is a way to get focused on direction not on magnitude.

upwardvector →

$$\boxed{\begin{array}{l} d = \dfrac{w^T P_1}{\|w\|} \\ d = \|w\| \cdot \|P_1\| \cdot \cos\theta \end{array}}$$

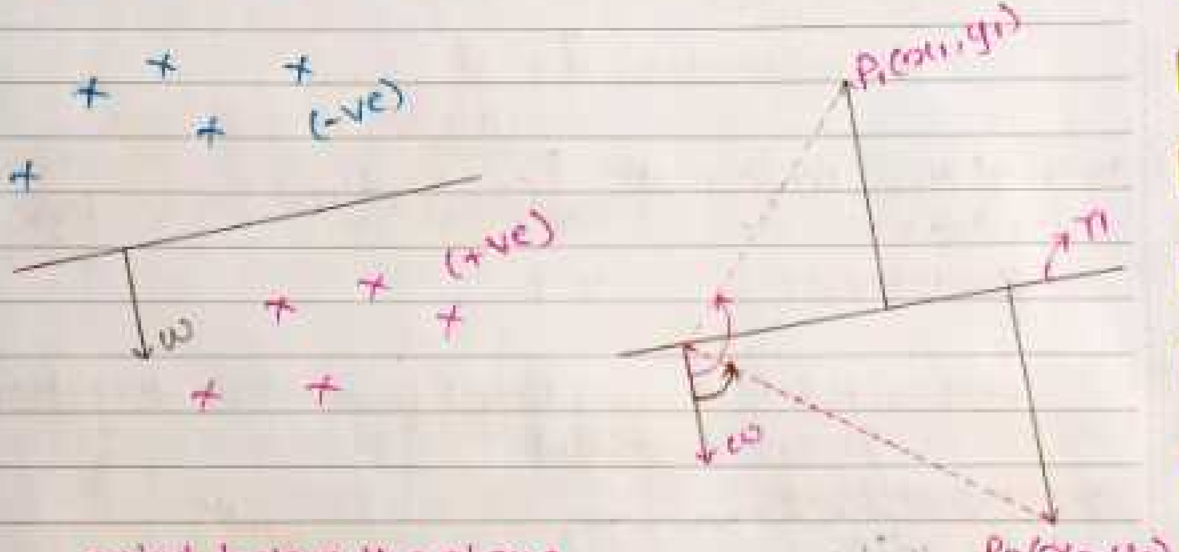


point above the plane
as $\theta < 90°$
$\cos\theta$ will always be +ve.

point below the plane, as $\theta > 90°$
$\cos\theta$ will always be -ve.

- If any point falling above the plane, then θ must be less than 90° (+value)
- If any point falling below the plane, then θ must be greater than 90°. (-ve value)

Downward vector:

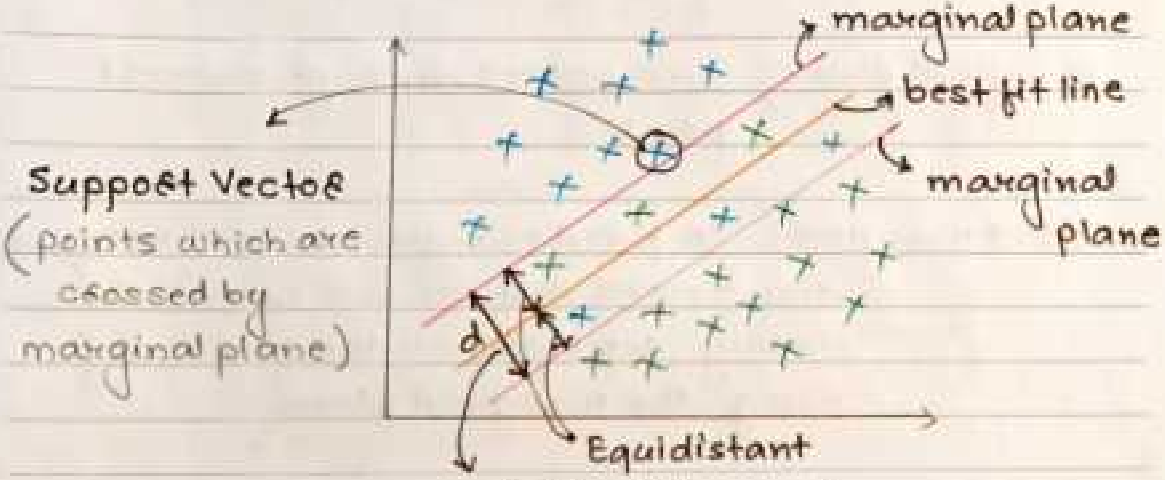


point below the plane,
θ < 90°
cosθ will always be '+ve'.

point above the plane, θ > 90°
cosθ will always be '-ve'.

Geometric Intuition Behind Support Vector Machine

Support Vector classifier (SVC)



d should be maximum

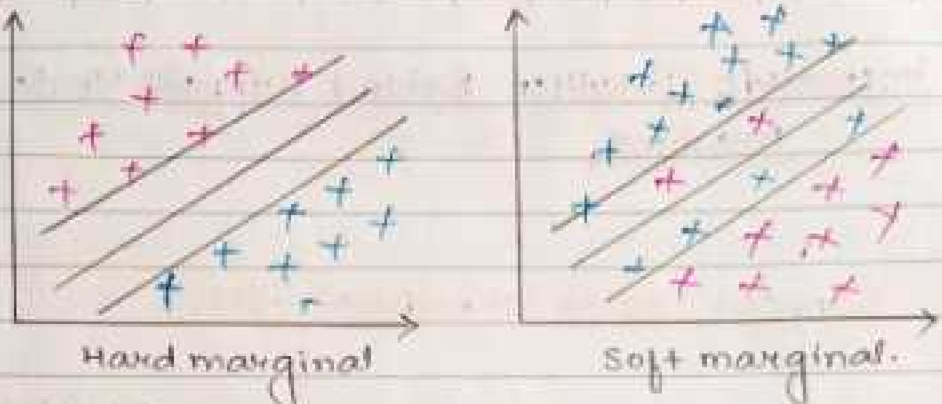
* you can have more than one support vector.

There will be many possible hyperplanes that separate different classes.

we have learnt in LR that the probability of a point belonging to any class given at very close to the hyperplane will be close to 0.5

So, we want a hyperplane that seperates (+ve) pts and (-ve) pts as far away as possible.

key idea of SVM. such hyperplane is called margin-maximizing plane.



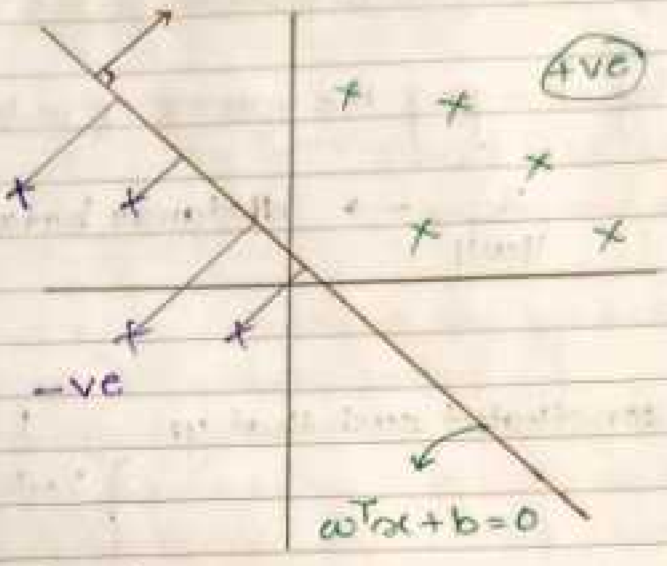
Marginal plane: Line passes through nearest points.

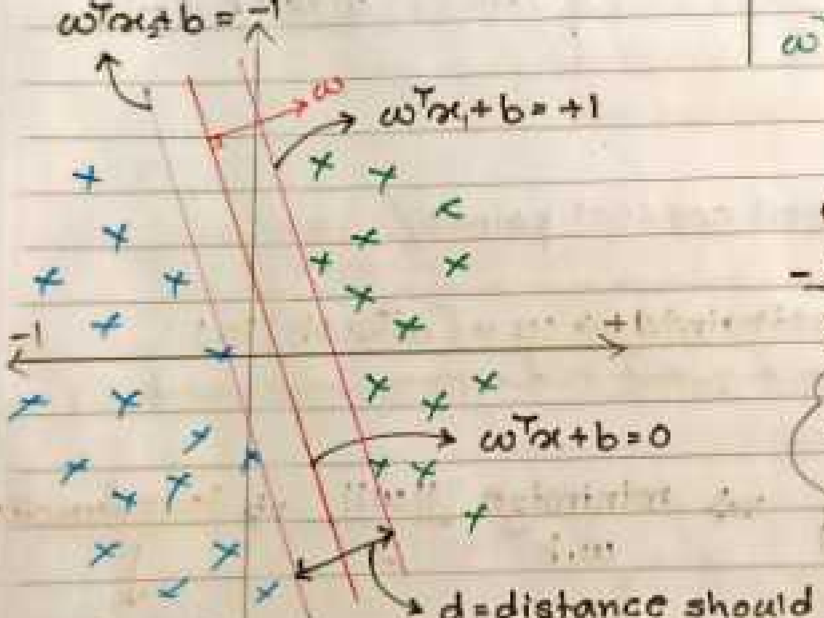
Hard marginal: In hard margin, we will not find any errors and we will be able to clearly separate all the points by using the marginal plane.

But, in real world there will be many overlapping with many errors, so marginal plane lines will be called soft margin.

→ marginal plane should be equidistance from best fit line.

SVM Mathematical Intuition:





$$\begin{aligned} \omega^T x_1 + \cancel{b} &= +1 \\ - \omega^T x_2 + \cancel{b} &= -1 \\ \hline \omega^T (x_1 - x_2) &= 2 \end{aligned}$$

$\therefore$ unit vector of ω,

$$\frac{\omega^T (x_1 - x_2)}{||\omega||} = \frac{2}{||\omega||}$$

$$\therefore \text{distance } (d) = \frac{\omega^T (x_1 - x_2)}{||\omega||} = \frac{2}{||\omega||}$$

Cost Function:

we have to maximize the value of $\frac{2}{||w||}$ by changing the values of w, b.

$\frac{2}{||w||} \longrightarrow$ distance between marginal plane.

constraint such that $y_i \begin{cases} 1 & w^T x + b \geq 1 \\ -1 & w^T x + b \leq -1 \end{cases}$

~~conditions~~

For all classified correct points,

$$\text{constraints} \rightarrow y_i \times (w^T x + b) \geq 1$$

$\underset{w,b}{\text{maximize}} \frac{2}{||w||} \Rightarrow \underset{w,b}{\text{minimize}} \frac{||w||}{2} \Rightarrow$ loss function $\downarrow$ minimize

* loss function focus on minimization.

Cost function:

minimize $\frac{\|w\|}{2}$ by changing w,b.

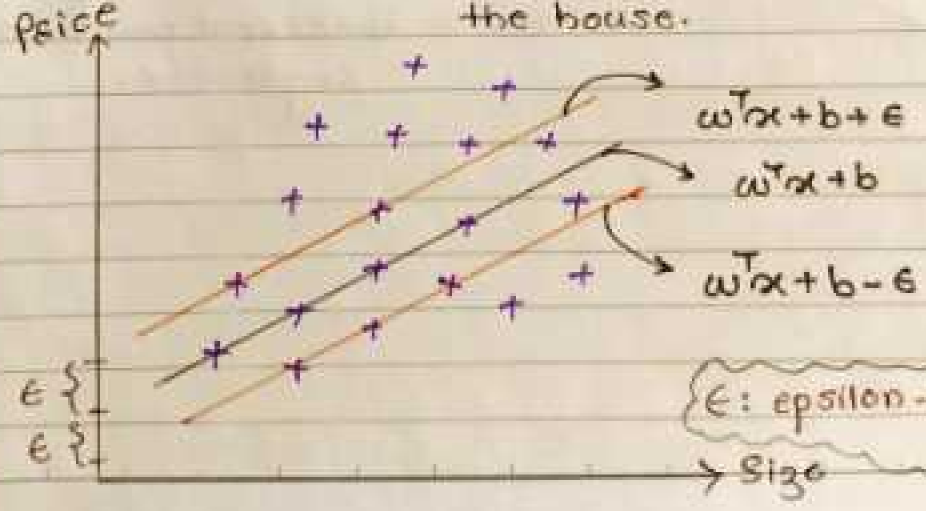
$$\boxed{\min \frac{\|w\|}{2} + \boxed{C_i \sum_{i=1}^{n} \xi_i}}$$ → Hinge loss for soft margin.

where, C_i : How many points we can ignore for mis-classification.
→ Hyperparameter

ξ_i (Eta): Summation of the distance of incorrect data points from the marginal plane.

Support Vector Regressor:

Problem Statement: Based on the size of the house, we have to predict price of the house.



ϵ : epsilon → marginal error

cost Function:

$$\underset{w,b}{\text{minimize}} \; \frac{\|w\|}{2} + \underbrace{C_i \sum_{i=1}^{w} \mathcal{E}_i}_{\text{Hinge loss}}$$

MAE

$$\boxed{\text{constraint: } |y_i - w_i x_i| \leq \epsilon + \mathcal{E}_i}$$

Eta gi

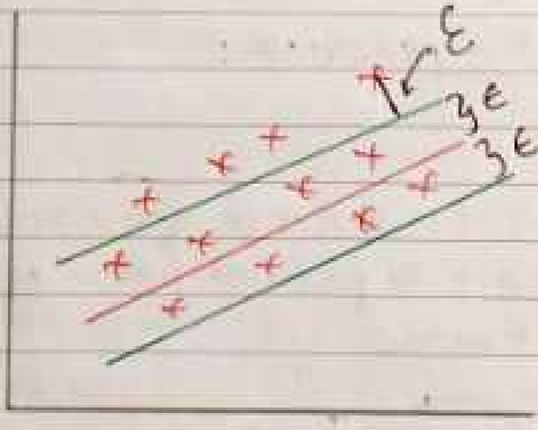
Truth point

predicted point

epsilon

ϵ : margin of error (to decide original plane)

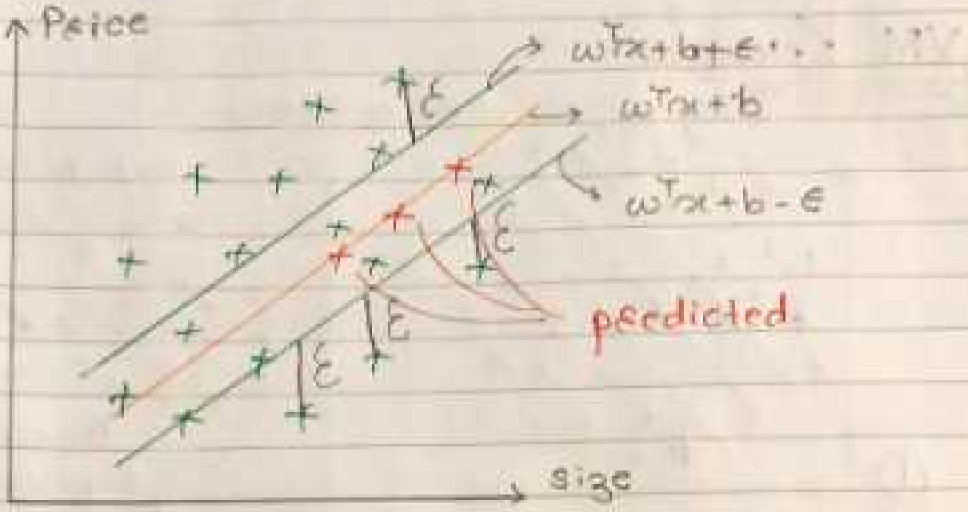
$\mathcal{E}_i$: error above the margin.



Hyperparameter:

→ keep adjusting ε to get best margin

→ we can't say incorrect point in regressor.



In Regressor, no complete incorrect value, because it will be continuous value.

Q.

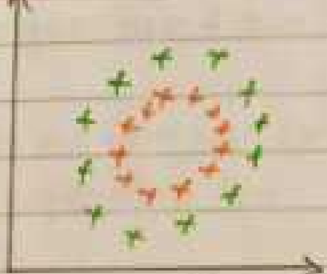
Is svm Impacted by the outliers?

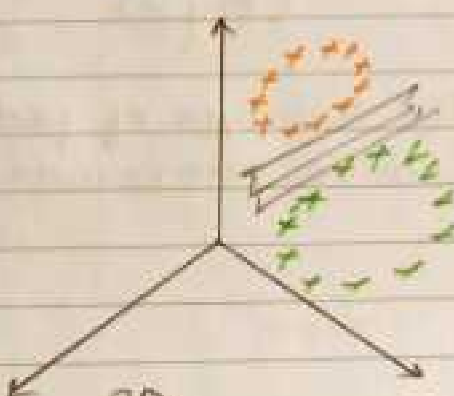
Yes, SVM is impacted by the outliers.

Does Standardization is need in svm?

Yes, we need to perform Normalization and standardization.

SVM kernel:



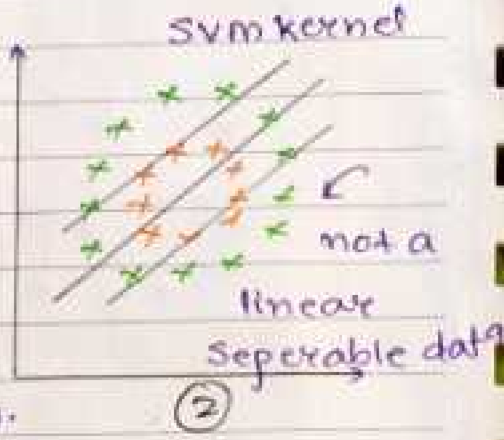


2D ⟶ 3D

kernels

SVM Kernels:

linear seperable data.



- when we create this (①) type of best fit line and marginal plane, we are actually solving the linear seperable data.

- → called as Linear SVC. (Fig①)

- If data is not a linear seperable data, you will not be able to create best fit line and not able to create a marginal plane even though we create it, the accuracy will be very low. (Fig 2)

-

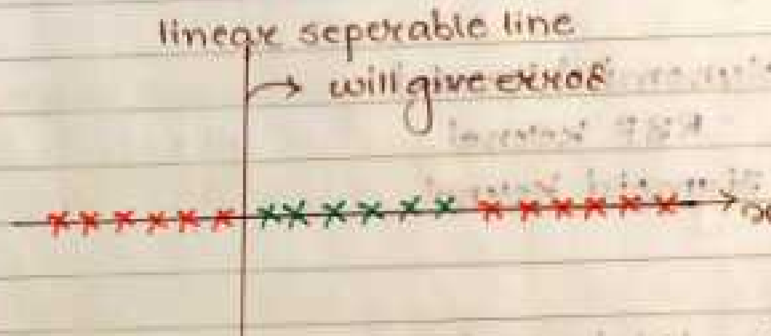
- For this type of problems, we have some more svm kernels.

what does svm kernels do?

→ The main aim is to apply some transformation technique. (some mathematical formula) on the dataset.

This transformation increases the dimension of the data.

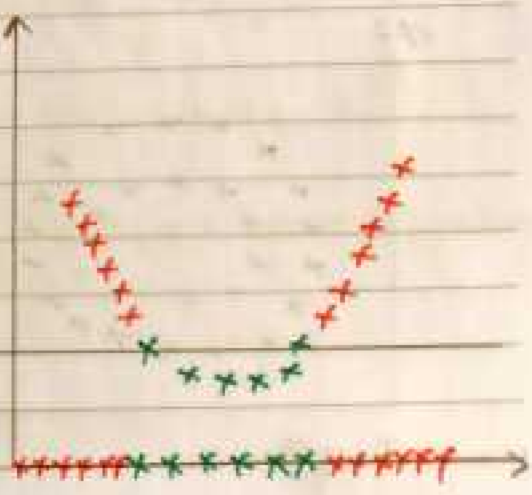
$$\boxed{\text{SVM kernels} \longrightarrow \overset{\text{(mathematical formula)}}{\text{Transformation}} \longrightarrow \text{Increasing the dimension of the data}}$$



∴ we will transform the data from 1D → 2D.

$\boxed{y = x^2}$

After →



Now, we can use linear seperable line.

$y = x^2$

so if $x = -7$ $y = 49$

$x = -3$ $y = 9$ and so on.

what is the advantage of doing this transformation?

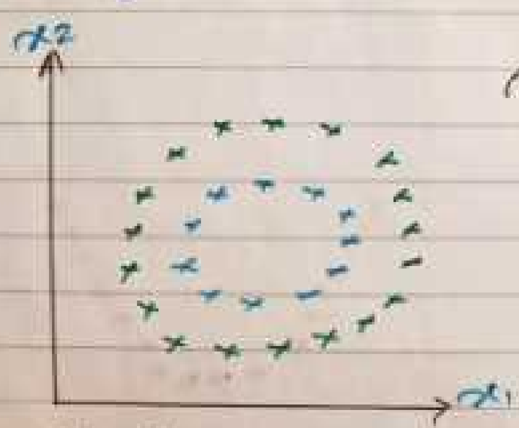
→ after transformation, we can apply linear svm or svc.

* when we ~~divide~~ convert 1D → 2D then we can divide all the points using single line which is called Linear svc.

Types of SVM kernel:

1. Polynomial kernel
2. RBF kernel
3. Sigmoid kernel

1. Polynomial kernel



→ not seperable with best fit line.

so, we need to convert 2D → 3D.

Fig 3.

Our main aim was to increase dimension,
2D $\longrightarrow$ 3D
so, hyperplane is created.

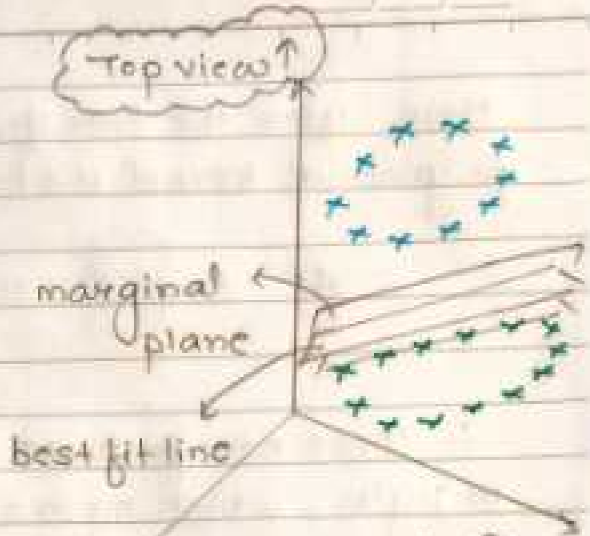


Fig ④.

Formula for Polynomial kernel:

$$\boxed{f(x_1, x_2) = \left(x_1^T x_2 + 1\right)^d}$$

d = dimension

If we are converting 2D $\rightarrow$ 3D, the value of $\underline{\underline{d=3}}$.

$$\therefore \quad x_1^T x_2 = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \cdot \begin{bmatrix} x_1 & x_2 \end{bmatrix}$$

$$= \begin{bmatrix} x_1^2 & x_1 \cdot x_2 \\ x_1 \cdot x_2 & x_2^2 \end{bmatrix}$$

3 unique values: $x_1^2, x_1 \cdot x_2, x_2^2$

Now, initially at the time of Fig 3, we have 3 features
$x_1 \quad x_2 \quad y$

Now, after the transformation / formula of polynomial kernel we have 6 features

$$x_1 \quad x_2 \quad \underbrace{x_1^2 \quad x_1 \cdot x_2 \quad x_2^2} \quad y$$

∴ These new features can be plotted as the 3D in fig 4., which means that

in fig 4, x_1 will be x_1^2

x_2 will be x_2^2

z will be $x_1 \cdot x_2$

and once we have all these points, we will be able to clearly seperate the points.

→ use polynomial kernel, to get better accuracy.

② Radial Basis Function Kernel (RBF kernel)

$$K(\vec{x}, \vec{l_i}) = e^{-\frac{\|\vec{x} - \vec{l_i}\|}{2\sigma^2}}$$

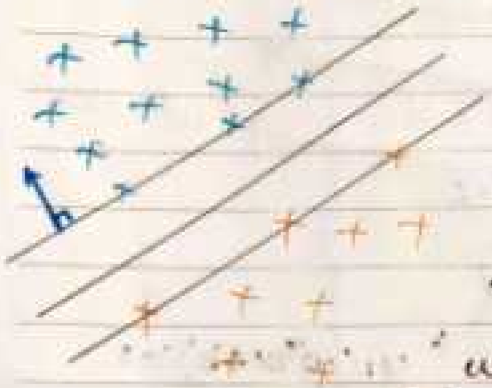
hyperparameter

③ Sigmoid kernel

It can be used as the proxy for neural networks.

$$K(x, x_i) = \tanh\left(\sigma x^T x_i + r\right)$$

Mathematical formulation of SVM.



Objective: To find a hyper-plane that does margin maximization.

we need to find a hyperplane π : margin-maximization

π : $w^T x + b = 0$ (w not necessarily unit vector).

π_+ : $w^T x + b = 1$

π_- : $w^T x + b = -1$

By simple co-ordinate concepts we can get

$$d = \frac{2}{\|w\|}$$

So, we have to maximize 'd' with the constraint that all (+ve) points will lie above π_+ & all (-ve) pts below π_-.

So, the optimization function ~~been~~ becomes

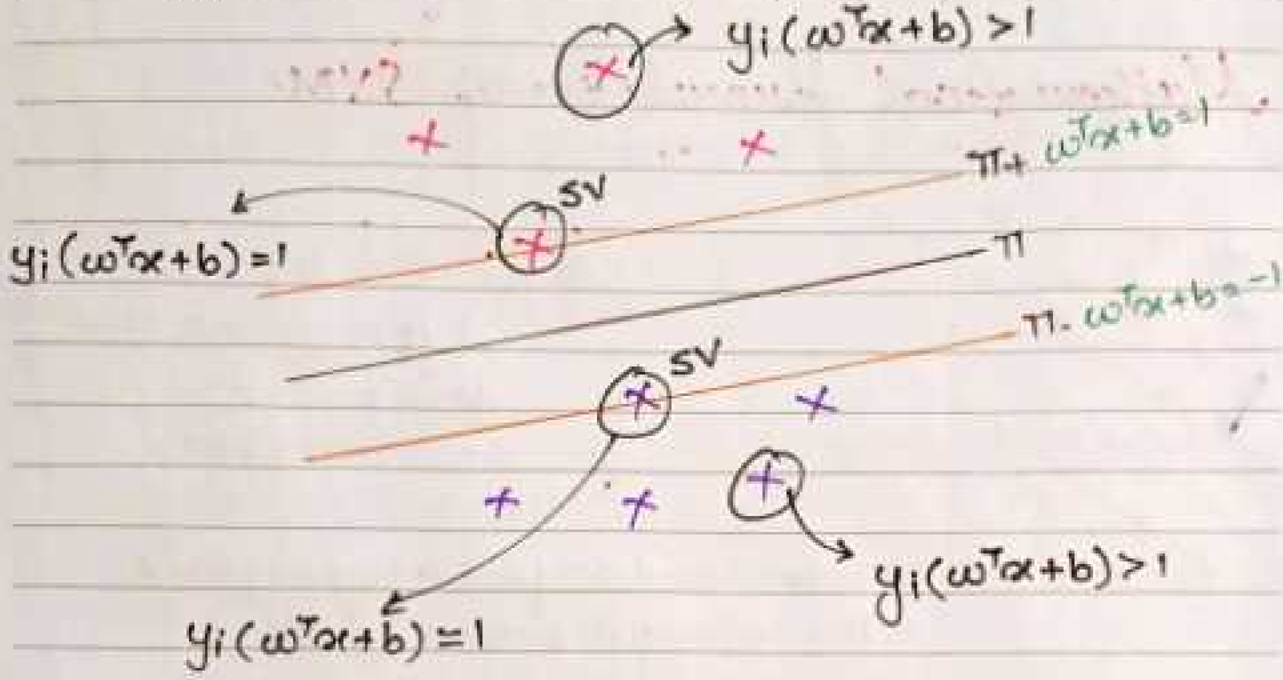
Hard margin

$$(w^*, b^*) = \underset{w,b}{\arg\max} \frac{2}{\|w\|}$$

s.t.

$$\forall i, \quad y_i(w^T x_i + b) \geq 1$$

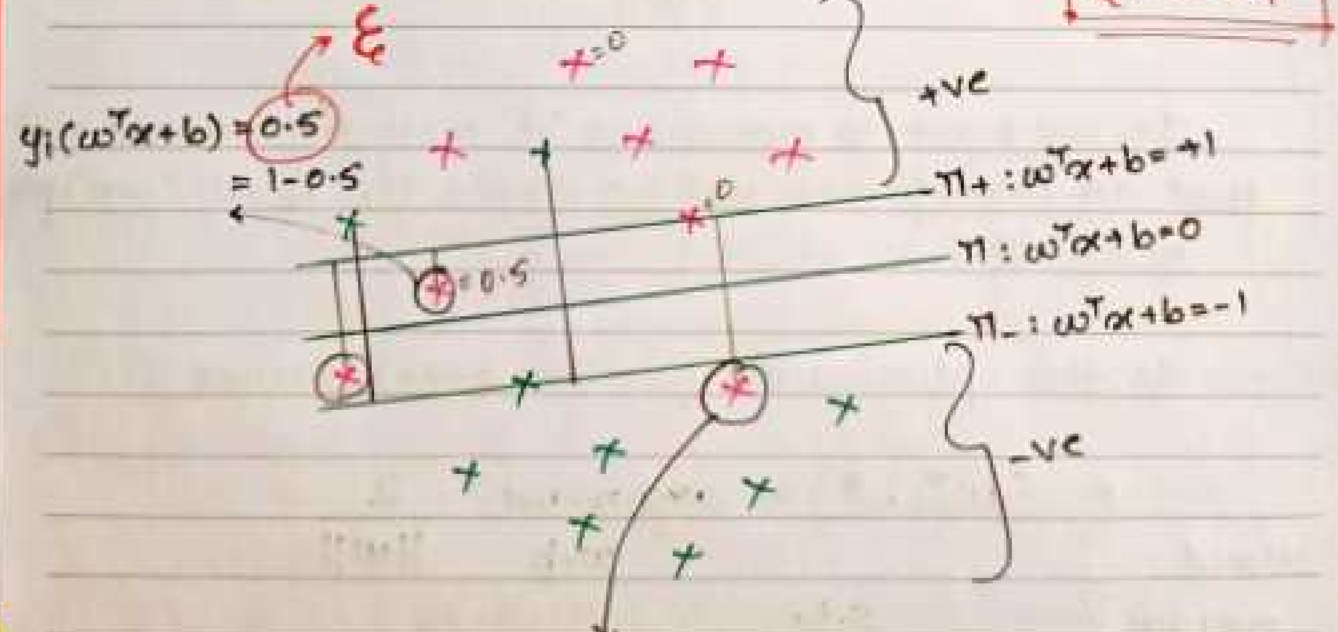
SV: support vector



But what if we have a dataset that's not linearly separable but almost linearly seperable?

→ we formulate an alternative soft-margin constraint that allows some error to persist.

$\boxed{\xi = \text{zeta}}$



$y_i(\omega^T x+b)=1.5$

i.e $= 1-(2.5)$
$=1.5$

$x_i \rightarrow \xi_i$ ξ = zeta

$\xi_i = 0$ if $y_i(w^T x + b) \geq 1$
(i.e. for correctly classified points)

$\xi_i > 0$ and it is equal to some unit of dist^n away from the correct hyperplane in the incorrect direction.

Now, we want to minimize these errors.

So, our optimization function becomes

$$\underset{w,b}{\arg\min} \frac{\|w\|}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i$$

$\frac{1}{\text{margin}}$

avg. distance of misclassified points (Hinge loss).

s.t.

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

def^n of zeta

C → hyperparameter

C↑ → tendency to make mistakes↓
→ overfit (High variance)

C↓ → underfit (High Bias)

$\forall i$ correctly classified points $\xi_i = 0$

incorrectly classified pts $\xi_i > 0$

maximizing $\frac{2}{||w||}$ same as minimizing $\frac{||w||}{2}$

soft margin of SVM:

$$(w^*, b^*) = \underset{w,b}{\arg\min} \frac{||w||}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i$$

s.t.

$$\left.\begin{aligned} y_i(w^T x + b) &\geq 1 - \xi_i \quad \forall i \\ \xi_i &\geq 0 \end{aligned}\right\} \text{constraints}$$

$x_i \to \xi_i$

ξ_i

ξ_i

away from the correct hyperplane in the incorrect direction.

Now, we want to minimize these errors.

So, our optimization function becomes

$$\underset{w,b}{\arg\min} \frac{||w||}{2} + C \cdot \frac{1}{n} \sum_{i=1}^{n} \xi_i$$

$\frac{1}{\text{margin}}$

avg. distance of misclassified points (Hinge loss).

defn of zeta

s.t.

$$y_i(w^T x_i + b) \geq 1 - \xi_i$$

$$\xi_i \geq 0$$

$\forall i$

C → hyperparameter

C↑ → tendency to make mistakes↓

→ overfit (High variance)

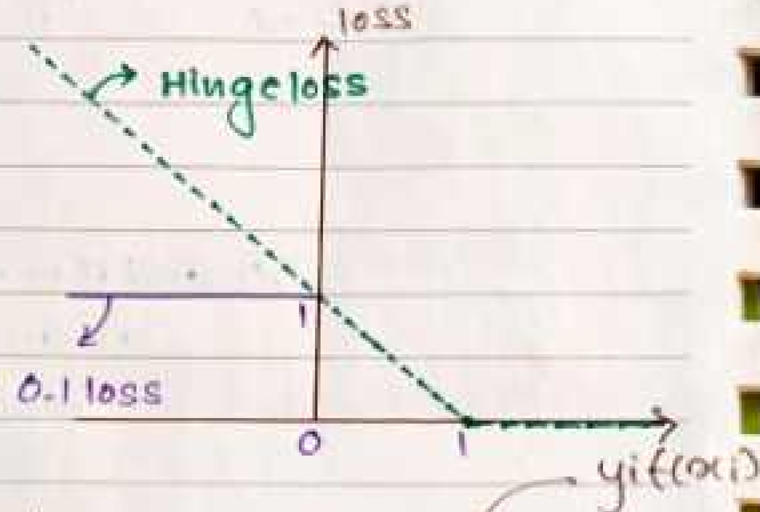
C↓ → underfit (High Bias)

correctly classified points $\xi_i = 0$

incorrectly classified pts $\xi_i > 0$

Loss minimization : Hinge Loss

Hinge loss + Regularization
gives us
svm.



although, Hinge loss also
not differentiable at $z_i = 1$,
but since it is continuous unlike
0-1 loss, we can work around
hacks to work with it.

$y_i f(x_i) =$
$y_i(w^T x + b)$
$= z_i$

$z_i > 0$: x_i is correctly classified
$z_i < 0$: x_i is incorrectly classified.

$$\text{Hinge loss} = \begin{cases} z_i \geq 1 \ ; & \text{Hing loss} = 0 \\ z_i < 1 \ ; & \text{Hinge loss} = 1 - z_i \end{cases}$$

alternatively,

$$\text{Hinge loss} = \max(0, 1 - z_i)$$

— Andrew Ng

Lagrange Duality

consider a problem of the following form:

$$\min_{\omega} \; f(\omega)$$
$$\text{s.t.} \; h_i(\omega)=0, \; i=1,\ldots,\ell.$$

we define the Lagrangian to be

$$\mathcal{L}(\omega,\beta) = f(\omega) + \sum_{i=1}^{\ell} \beta_i h_i(\omega)$$

Here, β_i's = Lagrange multipliers.
To find, set $\mathcal{L}$'s partial derivatives to 0;

$$\frac{\partial \mathcal{L}}{\partial \omega_i} = 0; \quad \frac{\partial \mathcal{L}}{\partial \beta_i} = 0,$$

and solve for ω and β.

consider the following, called primal optimization problem:

$$\boxed{\begin{array}{ll} \min_{\omega} & f(\omega) \\ \text{s.t.} & g_i(\omega) \leq 0, \; i=1,\ldots,k \\ & h_i(\omega) \leq 0, \; i=1,\ldots,\ell. \end{array}}$$

To solve it, we start by defining

generalized Lagrangian.

$$\mathcal{L}(\omega,\alpha,\beta) = f(\omega) + \sum_{i=1}^{k} \alpha_i g_i(\omega) + \sum_{i=1}^{l} \beta_i h_i(\omega)$$

Here, the α_i's and β_i's are the Langrange multipliers.

consider the quantity,

$$\theta_P(\omega) = \max_{\alpha,\beta;\alpha_i \geq 0} \mathcal{L}(\omega,\alpha,\beta)$$

Here, P = "primal".

Let some ω be given. If ω violates any of the primal ~~constraintsis~~ constraints

(i.e. if either $g_i(\omega) > 0$ or $h_i(\omega) \neq 0$ for some i)

then,

$$\theta_P(\omega) = \max_{\alpha,\beta;\alpha_i \geq 0} f(\omega) + \sum_{i=1}^{k} \alpha_i g_i(\omega) + \sum_{i=1}^{l} \beta_i h_i(\omega)$$

$$= \infty \quad \text{(1)}$$

Conversely, if the constraints are indeed satisfied for a particular value of ω,

then $\theta_p(\omega) = f(\omega)$. Hence

$$\theta_p(\omega) = \begin{cases} f(\omega) & \text{if } \omega \text{ satisfies primal constraints} \\ \infty & \text{otherwise} \end{cases}$$

Hence, if we consider the minimization problem

$$\min_{\omega} \theta_p(\omega) = \min_{\omega} \max_{\alpha,\beta;\alpha_i \geq 0} \mathcal{L}(\omega,\alpha,\beta)$$

same as, primal problem.

Optimal value of the objective to be $p^* = \min_{\omega} \theta_p(\omega)$

value of the primal problem.

Now, we define

$$\theta_D(\alpha,\beta) = \min_{\omega} \mathcal{L}(\omega,\alpha,\beta)$$

where, D = "dual"

Note: In defn θ_p we were optimizing (maximizing) w.r.t α, β here we are minimizing w.r.t. ω.

∴ Dual optimization problem:

$$\max_{\alpha,\beta:\alpha_i \geq 0} \theta_D(\alpha,\beta) = \max_{\alpha,\beta:\alpha_i \geq 0} \min_{\omega} \mathcal{L}(\omega,\alpha,\beta)$$

Optimal value of the dual problem's objective :

$$d^* = \max_{\alpha,\beta:\alpha_i \geq 0} \theta_D(\omega)$$

How are the primal and the dual problems related ?

$$d^* = \max_{\alpha,\beta:\alpha_i \geq 0} \min_{\omega} \mathcal{L}(\omega,\alpha,\beta) \leq \min_{\omega} \max_{\alpha,\beta:\alpha_i \geq 0} \mathcal{L}(\omega,\alpha,\beta)$$
$$= p^*$$

always remember :

"~~min~~ max min" of a function always being less than or equal to the "min max".

for certain conditions,

$$d^* = p^*$$

i.e. there exists a_i, b_i, so that
$h_i(\omega) = a_i^T\omega + b_i$

affine → linear, except contains
extra intercept term b_i

Suppose f and the g_i's are convex, and the h_i's are affine.

Suppose that the constraints g_i are feasible; means there exists some ω so that $g_i(\omega) < 0$ for all i.

under above assumptions, there must exist $\omega^*, \alpha^*, \beta^*$ so that ω^* is the solution to the primal problem α^*, β^* are the solution to the dual problem, and moreover $p^* = d^* = \mathcal{L}(\omega^*, \alpha^*, \beta^*)$

moreover, ω^*, α^* and β^* satisfy the Karush-Kuhn-Tucker (KKT) conditions

$$\frac{\partial}{\partial \omega_i} \mathcal{L}(\omega^*, \alpha^*, \beta^*) = 0, \quad i = 1, \ldots, d \quad \text{③}$$

$$\frac{\partial}{\partial \beta_i} \mathcal{L}(\omega^*, \alpha^*, \beta^*) = 0, \quad i = 1, \ldots, l \quad \text{④}$$

$$\alpha_i^* g_i(\omega^*) = 0, \quad i = 1, \ldots, k \quad \text{⑤}$$

$$g_i(\omega^*) \leq 0, \quad i = 1, \ldots, k \quad \text{⑥}$$

$$\alpha^* \geq 0, \quad i = 1, \ldots, k \quad \text{⑦}$$

moreover, if some $\omega^*, \alpha^*, \beta^*$ satisfy the KKT conditions, then it is also a solution to the primal and dual problems.

Eqn ⑤, which is called the KKT dual complementarity condition.

Specifically, it implies that if $\alpha_i^* > 0$, then $g_i(w^*) = 0$.

i.e. "$g_i(w) \leq 0$" constraint is active, meaning it holds with equality rather than with inequality.

This will be key for showing that svm has only a small number of "support vectors".

Dual Form of SVM

Following (primal) optimization problem for finding the optimal margin classifier:

$$\min_{\omega, b} \frac{1}{2} \|\omega\|^2$$

s.t.

$$y_i(\omega^T x_i + b) \geq 1, \; i = 1, \dots, n \quad \text{— (8)}$$

constraints as

$$g_i(\omega) = -y_i(\omega^T x_i + b) + 1 \leq 0$$

when we construct ~~Lang~~ Lagrangian for optimization problem we have:

$$\boxed{\mathcal{L}(\omega, b, \alpha) = \frac{1}{2}\|\omega\|^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(\omega^T x_i + b) - 1 \right]} \quad \text{— (9)}$$

Note: there's only "α_i" but no "β_i" Lagrange multipliers, since the problem has only inequality constraints.

To find the dual form, first minimize $\mathcal{L}(\omega,b,\alpha)$ w.r.t ω and b (for fixed α) to get θ_D.

$$\nabla_\omega \mathcal{L}(\omega,b,\alpha) = \omega - \sum_{i=1}^{n} \alpha_i y_i x_i = 0$$

$$\therefore \boxed{\omega = \sum_{i=1}^{n} \alpha_i y_i x_i} \quad \text{—— (10)}$$

derivative w.r.t b,

$$\frac{\partial}{\partial b} \mathcal{L}(\omega,b,\alpha) = \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \text{—— (11)}$$

If we take the defn of ω in eqn (10) and put it into the Lagrangian eqn (9), we get

$$\mathcal{L}(\omega,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j (x_i)^T x_j - b \sum_{i=1}^{n} \alpha_i y_i$$

but from eqn (11), last term = 0

$$\therefore \mathcal{L}(\omega,b,\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j x_i^T x_j$$

∴ with the constraint $\alpha_i \geq 0$, we obtain

dual optimization problem

$$\max_{\alpha} W(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} y_i y_j \alpha_i \alpha_j \langle x_i, x_j \rangle$$

$$\text{s.t. } \alpha_i \geq 0, \quad i = 1, \ldots, n$$

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

Important Observations:

1. for every x_i, we have an α_i
2. x_i's only occur in the form of $x_i^T x_j$

on solving we get

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

for non-SV, $\alpha_i = 0$
so, for $f(q) \rightarrow$ only the support vector matters.

3. $\alpha_i > 0$ only for support vectors else 0.

since x_i always occurs only as $x_i^T x_j \rightarrow x_i \cdot x_j$

cosine_sim $\langle x_i, x_j \rangle$

so, basically cosine similarity value are only required to solve the optimization and ultimately get the model.

we can replace cosine-similarity with any other kind of similarity which makes it more powerful.

Generalized dual form-

$$\max_{\alpha_i} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \underbrace{k(x_i, x_j)}_{\text{called Kernel function.}} \ \ldots\ldots$$

(can be any kind of similarity b/w x_i & x_j)

Even during evaluation x_i occurs only as $x_i^T x_q$

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

Kernel trick → replacing $x_i^T x_j$ with generalized similarity function $k(x_i, x_j)$.

Quadratic Programming.

The hard margin and soft margin problems are both convex quadratic optimization problems with linear constraints.

Such problems are known as Quadratic Programming (QP) problems