- Understanding Machine Learning Algorithms: Handwritten Notes -

Table of Contents

Notes by RaviTeja G

10. Exploring Step 3 - Train Model on Dataset
    - Types of Learning
    - Under Fitting and OverFitting
    - Regularization techniques
    - Hyperparameter Tuning
11. Exploring Step 4 - Evaluation of a Model
    - Evaluation Metrics
    - Confusion Matrix
    - Recall/Sensitivity
    - Precision
    - Specificity
    - F1 Score
    - AUC and ROC Curve
    - Analysis of a Model
12. Supervised Learning
    - Linear Regression
    - Regularization Techniques
    - Logistic Regression
    - Decision Trees
    - Ensemble Techniques
    - Random Forests
    - AdaBoost
    - Gradient Boost
    - XG Boost
    - K-Nearest Neighbours
    - Support Vector Machines
    - Naive Bayes Classifiers
13. Unsupervised Learning
    - Clustering Techniques
    - K-Means Clustering
    - Hierarchical Clustering
    - DB Scan Clustering
    - Evaluation of Clustering Models
    - Curse of Dimensionality
    - Principal Component Analysis
14. Cheat Sheet of Supervised and Unsupervised Algorithms

Machine Learning

Introduction:-

→ At a high-level, machine learning is simply the study of teaching a computer program/ algorithm how to progressively improve upon a set task that it is given.

So, more practically it is the study of how to build application that exhibit this, iterative improvement.

there are many ways to implement it and largely there are 3 major categories.

1) Supervised Learning.

2) unsupervised Learning.

3) Reinforcement Learning.

And also there is semi-supervised learning.

machine learning algorithms are the 'engine of machine learning', meaning it is the algorithms that turn a dataset into a model.

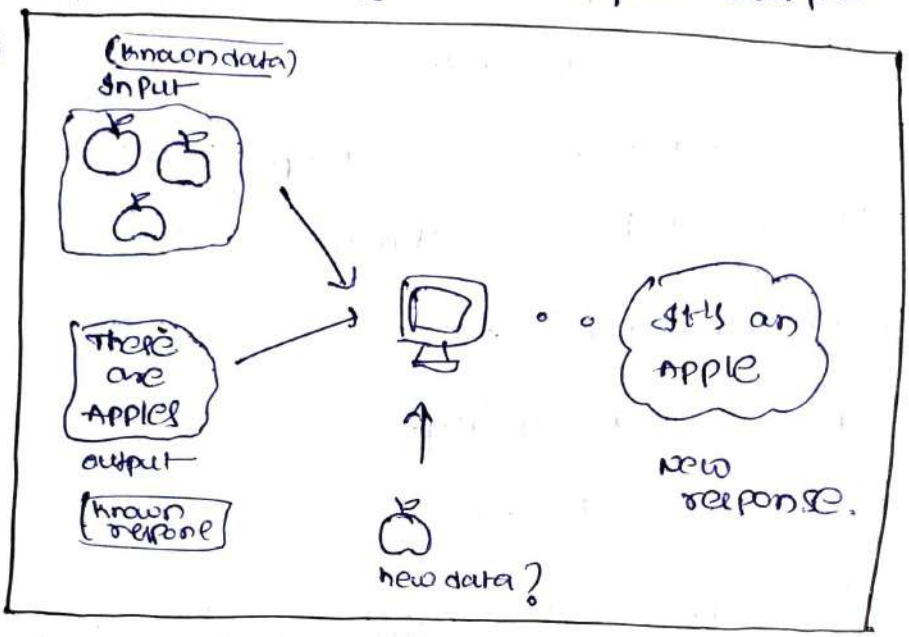
→ which kind of algorithm works best and what to choose (supervised, unsupervised, classification, regression, etc..) depends on the kind of problem you're solving, the computing resources available & the nature of the data

Briefing the types:-

(1) Supervised learning

Supervised learning is the most popular paradigm for performing machine learning operations. It is widely used for data where there is a precise mapping b/w input-output data.

i.e;



→ Given data in the form of examples with labels, we can feed a learning algorithm these 'example-label' pairs one by one. overtime, the algorithm will learn to approximate the exact nature of the relationship b/w examples & their labels. (It's called training the data)

→ when fully trained, the supervised learning algorithm will be able to observe a new, never-before seen example & predict a good label for it.

→ And so supervised algorithms are called 'task-oriented'. As we provide it with more & more examples, it is able to learn more & more properly so that it can undertake the task & yield us the output more accurately.

It is exhibited in many of following applicatns

Face Recognition:-

Face lock app in our phone has been using a supervised learning algorithm that it is trained to recognize your face. Having a system that takes a photo, finds faces & guesses who that is in the photo (suggesting a tag) is a supervised process.

Spam classifications:-

Spam filter is a supervised learning system. Fed email examples & labels (spam/not-spam) these systems learn how to filter out malicious emails so that the user is not harassed.

Advertisement popularity:-

Selecting advertisements that will perform well is often a supervised learning task. Many of ads you see as you browse the are placed there because a learning algorithm said that they would be reasonable popularity (more clicked).

and many examples... (3)

Types of Supervised learning

1) Regression
2) classification
3) neural networks
4) Boosting techniques.

→

Regression

→ Regression algorithms predicts a '**continuous value**' based on the input variables.

Regression algorithms:-

1) Linear Regression
2) Polynomial Regression
3) Exponential Regression

) Logarithmic Regression

Applications:-

① Risk Assesment
② score prediction etc-

④ ——|—— Temp predict

classification

→ we use classification algorithms for predict set of items class or category.

classification algorithms:-

Logistic Regression
1) K-nearest neighbours
2) Decision trees
3) Random forest
4) support vector machine (svm)
5) Naive Bayes
6) Ada boost
7) XG boost

Applications:-

① Fraud detection
② email spam detection
③ Image classification
④ Diagnostics etc...

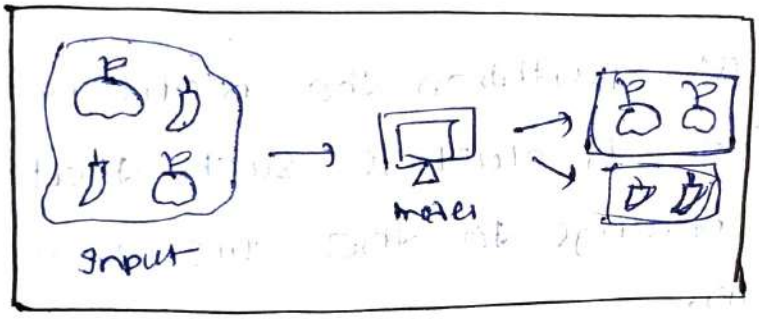
Hot | cold

② unsupervised learning

[Based on Hebbian learning]

→ To easy understand, in case of unsupervised learning algorithm, the data is not explicitly labeled into diff. datas, that is no labels. The model is able to learn from the data by finding implicit patterns.

→ unsupervised learning algorithms identify the data based on their density / structures, similar segments & other similar features.



Types of unsupervised learning :-

1) clustering
2) Dimensionality reduction & visualization
3) Anamomaly detection
4) Association
5) Auto encoders

1) clustering :-

Clustering, also known as cluster analysis, is a technique of grouping similar sets of objects in the same group that is diff from the objects in other group.

→ Some of the essential clustering techniques are as follows:-

a) K-means

In this we partition the n observations in the data into K cluster such that each observation belongs to the cluster with nearest mean.

b) DBSCAN

It groups the data based on density. It groups together the points that are given in space & mark the outliers in low density regions.

c) Hierarchical clustering

A hierarchy of cluster is built.

d) Mean Shift

2) Dimensionality Reduction & visualization:

a) Principal Component Analysis (PCA):

Reduce data from more dimensions to lower dimensions while attempting to preserve the variance.

→ we Reduce the size of data to extract useful information.

b) t-Distributed Stochastic Neighbor Embedding (t-SNE):

It is used to visualize high-dimensionality data in a 2D / 3D space.

3) Anomaly detection:

Anomaly detection techniques detect outliers in the unlabeled data under an assumption that most of the data examples are normal. by observing the instances that fit the remainder dataset.

one-class classification:

Train a model on only one-class, if anything lays outside of this class, it may be an anomaly.

Algorithms for doing so include..

(i) one-class k-means
(ii) one-class SVM
(iii) Isolation forest

4) Association

We use association algorithm for associating co-occurring items / events.

Association algorithms

- Apriori

5) Autoencoders :- (Dimensionality Reduction)

Autoencoders take input data, compress data into a code, then tried to recreate the input data from summarized code.

→ Autoencoders can remove noise from visual data like images, video / medical scans to improve quality.

Note :-

→ Autoencoders can also be used to find outliers (Anomaly).

③ Reinforcement Learning

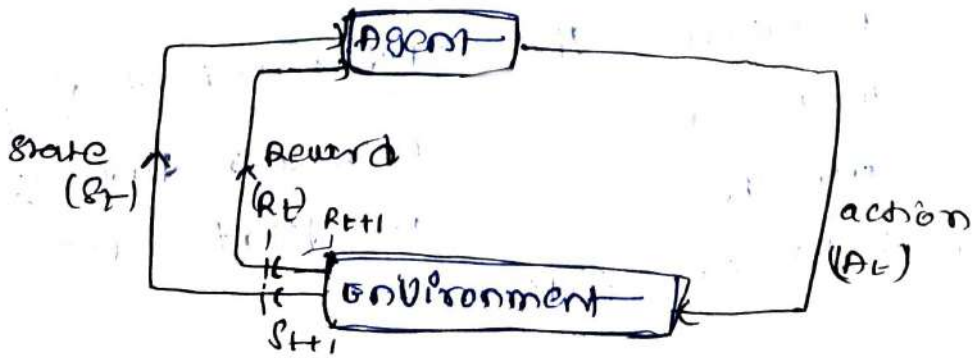
It doesn't require any data as labeled input or labeled output. All it has is the main data.

→ Like, trial & error method. If it fails in a trial then it learns & tries & it learns etc.

→ Games are a real example of RL.
Take a chess game model. All it has is the environment. First it takes a step & sees the outcomes & learns from it and so on.

→ In this kind of RL, AI agents are attempting to find the optimal way to accomplish a particular goal, or improve performance on specific task. As the agent takes action that goes toward the goal, it receives a reward.

overall aim: Predict the best next step to earn the best reward.



when there is no train dataset, it learns from experience.

④ Semi-supervised data

→ Semi-supervised learning is, for the most part, just what it sounds like: a training dataset with both labeled & unlabeled data.

→ this method is particularly useful when extracting relevant features from the data is difficult & labelling examples is a time-intensive task for experts.

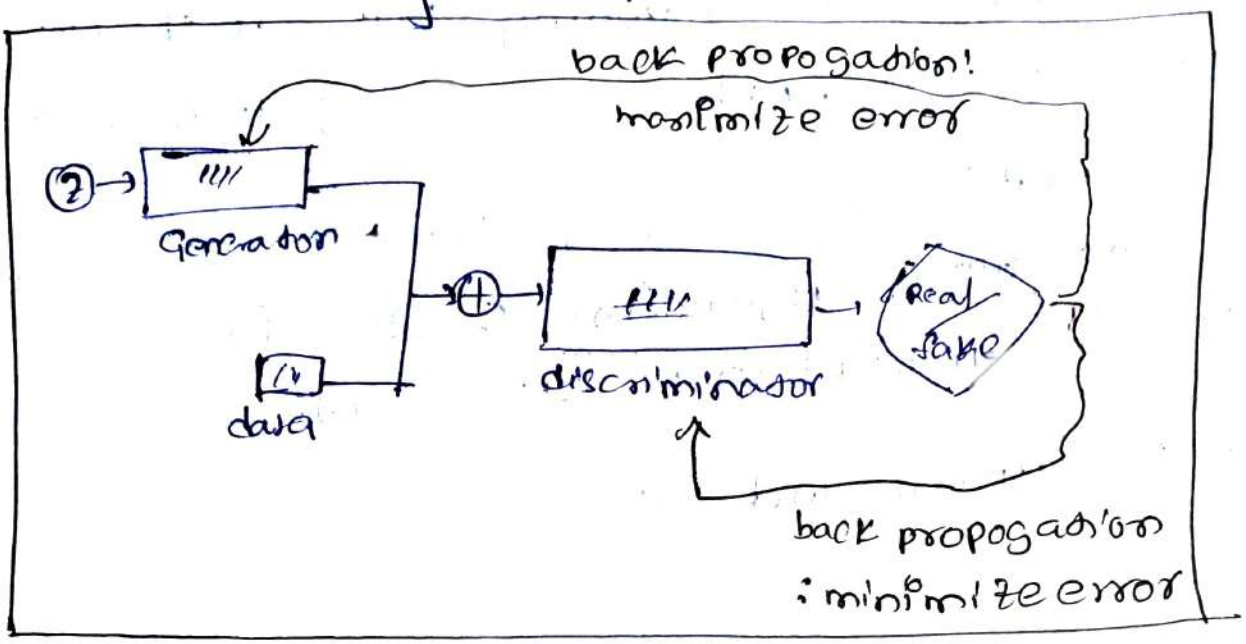
→ A popular training method that starts with a very small set of labelled data is using 'general adversarial networks' / GAN's

like two deep learning networks in competition, each trying to outsmart other. that's a GAN.

→ one of the networks called generator, tries to create new data points that mimic the training data.

→ the other network the discriminator, pulls in these newly generated data & evaluates whether they are part of training data or fake.

→ the networks improve in a positive feedback loop - as the discriminator gets better at separating the fakes from originals, the generator improves its ability to create convincing fakes.



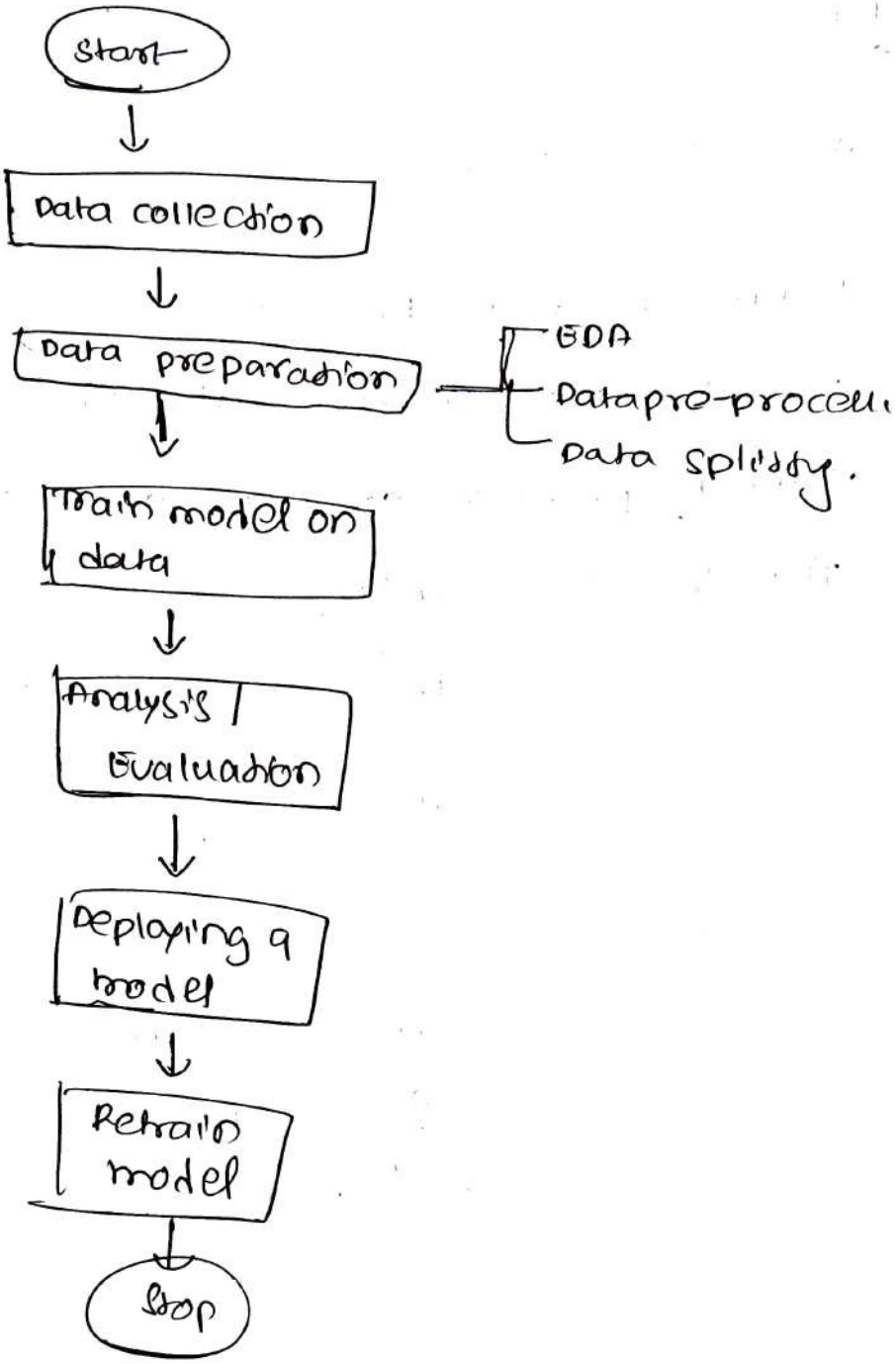
eg:-

→ common situations for this kind of learning are medical images like CT Scan (or) MRI's. A trained radiologist can go through & label a small subset of scans for tumors or diseases.

→ It would be too time-intensive & costly to manually label all the scans — but the deep learning network can still benefit from the small proportion of labeled data & improve its accuracy compared to fully unsupervised models.

→ It's the basic knowledge about algorithms.
→ Before going deep into algorithms.
better to know the flow chart of
machine learning. I feel ML is like a
christopher Nolan movie, first time you
don't understand anything but u should
keep watching, At last everything makes sense
Same works with ML projects, take a small
prof on a algorithm & see it.
you will understand the path.
So, let's understand the path.

ML Road map (Steps in machine learning project)



(*) understanding the process :-

Step (1)

Data collection

a) Questions to ask ?

1) what kind of problem are we trying to solve?
2) what data sources already exist?
3) what privacy concerns are there?
4) is the data public?
5) where should we store the data?

b) understanding the types of data

1) Structured data ?

Data which appear to be tabulated format can contain diff types of data.

like
i) nominal / categorical.
ii) numerical
iii) ordinal
iv) time series

2) unstructured data?

Data with no rigid structure.

(Images, Video, natural language text, speech)

Step (2)

Data preparation

It has 3 main process.

a) Exploratory data analysis (EDA), understanding data.

b) Data preprocessing, preparing your data to be modelling

c) Data splitting.

---

a) EDA, understanding the data

1) what are feature variables (input) and the target variables (output)?

Eg: For predicting heart diseases, feature variable may be a person's age, weight, heart rate etc.
And target variable is whether or not they have heart disease

2) what kind of data do you have?
Structured, unstructured, categorical / numerical.

3) Are there missing values? should you remove them / fill them with feature imputation?

4) what are outliers?
How many of them are there?
Are they out by much (3+SD)?
Why are they there?

5) Are there questions you could ask a domain expert about the data?

b) data preprocessing, preparing your data to be modelled

① Feature imputation:-

(Filling missing values)

→ A ML model can't learn on data that isn't there so,

i) single imputation: Fill with mean/median of column

ii) multiple imputation:- model other missing values and fill with what your model finds.

iii) KNN (k-nearest neighbour) :- Fill data with a value from another example which is similar.

iv) many more such as random imputation, last observation carried forward (for time series), moving window, most frequent.

∴ there are ways to handle missing values.

② Feature encoding:-

(Turning values into numbers)

→ A ML model requires all values to be numeric

i) OneHotEncoding:-

Turn all unique values into list of 0's & 1's where the target value is 1 & the rest are 0's.

Eg:- car colors green, red, blue,

a green car's color feature would be [1,0,0]

& a red one would be [0,1,0]

& a blue one [0,0,1].

ii) LabelEncoder :-

Turn labels into distinct numerical values

Eg:
If your target variables are diff animals as dog, cat, bird then 0, 1, 2 respectively

iii) Embedding encoding :-

Learn a representation amongst all the diff data points.

Eg:
A language model is a representation of how diff words relate to each other. Embeddings are also becoming more widely available for structured data.

---

(3) Feature normalization (scaling) or standardization :-

When your numerical variables are on diff scales. (like size of land b/w 1000 - 20000 sq.ft).
In such case some ML models don't perform well. Scaling / standardization help to fix this.

i) Feature scaling (normalization) :-

Shifts your values so they always appear b/w 0 & 1.

* This is done by subtracting min value & divide with range. (value b/w 0 & 1 but numerical relationship is still there)

ii) Feature standardization :-

Standardized all values so they have a mean of 0 & variance of 1.

* this is done by subtracting mean & divide with SD.
* values doesn't end up b/w 0-1 (if variable).
* Standardization is more robust to outliers than scaling.

④ Feature Engineering :-

→ Transform data into potentially more meaningful representations by adding in domain knowledge.

i) Decompose

Like, turn a date (such as 2020-06-18 to 20:16: into hour-of-day, day-of-week, day-of-month, is-holi etc.

ii) Discretization

(Turning larger groups to smaller groups)

- For numerical variables,
  Like age you may want to move it into bucket such as over-50 (under-50, (21-30, 31-40 etc.) this process is known as (BINNING).
- For categorical variables,
  such as car color, this may mean combining colors such as (light-green), (dark-green), (lime green) into a single green

iii) Crossing & Interaction features

(combining two/more features)

→ depends on situation. Sometime adding / subtracting two features could help.

iv) Indicator features :-

like using is-paid-traffic for paid

[using other parts of data to indicate something potentially significa

→ create a x-missing feature for wherever a column x contains a missing value.

⑤ Feature selection :-

→ selecting the most valuable features of your dataset to model.

→ Potentially reducing overfitting & training time (less overall data & less redundant data to train on) & improving accuracy.

i) Dimensionality reduction :-

A common dimensionality reduction method, PCA takes a larger no. of dimension (features) & uses linear algebra to reduce them to less dimensions. For example, say you have 10 numerical features you could run PCA to reduce it to 3.

ii) Feature importance :-

Fit a model to a set of data, then inspect which features were most important to the results, remove the least important ones. (can use "TPOT" for this)

⑥ Dealing with imbalances :-

Does your data have 10,000 examples of one class, but only 100 examples of another?

→ collect more data (if possible)

→ use scikit-learn-contrib imbalanced-learn Package.

→ use SMOTE (synthetic minority over-sampling technique) which creates synthetic samples of your minor class to try & level the playing field.

→ so this is all about Data Preprocessing.

c) Data splitting:

1) training set (usually 90-80% of data):
model learns on this.

2) validation set (typically 10-15% of data):
model hyperparameters are tuned on this.

3) Test set (usually 10-15%): (Dev set)
models final performance is evaluated on this
* Do not use this dataset to tune the model

Q. why shouldn't we tune on test set?

A.
To do the regularization we have to take 100 diff models & try on the test data. And so last we finds (suppose) the best hyperparameter that produces a model with low generalization error, say just 5%.

And you launch it to production & found it produces 15% error.

* Now, here is what happened as you trying every hyperparameter on the same test data eventually your model learned the test-data & fitted to it thus reducing generalization errors but not improving model.
this is called "overfitting"

→ So to avoid we take validation set / cross-validation

Step ③

Train model on dataset

It has 3 main process.

a) choose an algorithm
b) overfit the model
c) Reduce overfitting with Regularization.

---

a) Algorithm can be choosed by understanding supervised, unsupervised (explained previous)

b) Types of learning :-

i) Batch Learning :-

All of your data exists on a big static warehouse & you train a model on it. You may train a new model once per month once you get new data. Learning may take a while & isn't done often. Runs in production without learning (can be retrained).

ii) Online Learning :-

Your data is constantly being updated & you constantly train new models on it. Each learning step is usually fast & cheap.
Runs in production & learns continuously.

iii) Transfer learning:-

Take the knowledge one model has learned & use it with your own. Gives you the ability to leverage SOTA (state of the art) models for your own problems.
Helpful if you dont have much data / vast compute resources.

iv) Active learning:-

Also referred to as "human in the loop" learning. A human expert interacts with a model & provides updates to labels for samples which the model is most uncertain about.

v) Ensembling:-

Not really a form of learning, more combining algorithms which have already learned in some way to get better results.

Underfitting:-

Happens when your model doesn't perform as well as you'd like on your data. Try training for a longer & more advance model.

---

overfitting :-

Happens when your validation loss (how your model is performing on the validation dataset, lower is better) starts to increase. or if you don't have a validation set,

(*) Happens when the model performs far better on the "training set" than on the "test set".

(eg:- 99% accuracy on training set, 67% accuracy on test set) → overfitting the training set

Fix through various regularization techniques.

Regularization :- [techniques to prevent/reduce overfitting.]

i) L1 (Lasso) & L2 (ridge) regularization :-

→ L1 regularization sets unneeded feature coefficients to 0.

→ L2 contains a model's features (won't set them to 0)

ii) Dropout :-

Randomly remove parts of your model. So the rest of it has to become better.

iii) Early Stopping :-

Stop your model from training before the validation loss starts to increase too much / more generally, any other metric has stopped improving. Early stopping usually implemented in the form of a model callback.

iv) Data Augmentation :-

manipulate your dataset in artificial ways to make it harder to learn.

→ For example, if you're dealing with images, randomly rotate, skew, flip & adjust the height of your images.

→ this makes your model have to learn similar patterns across diff styles of the same image (harder).

→ use functions like ImageDataGenerator in keras or transforms in torchvision.

v) Batch Normalization :-

Standardize inputs as well as adding two parameters (beta, how much to offset the parameter for each layer & epsilon to avoid division by zero) before they go into next layer this often results in faster training speed since the optimizer has less parameters to update.

maybe a replacement for dropout in some networks.

Hyperparameter Tuning

→ Run a bunch of experiments with diff model settings & see which works best.

a) Setting a learning rate:-

(often the most important hyperparameter)

Generally

High learning rate = algorithm rapidly adapts to new data

Low learning rate = algorithm adapts slower to new data (est- for transfer learning)

i) Finding the optimal learning rate:-

Train the model for a few hundred iterations starting with a very low learning rate (10^{-6}) & slowly increase it to very large value (e^{1}) then plot the loss vs learning rate (using a log-scale for learning rate), you should see a U-shaped curve, the optimal learning rate is about 1-2 notches to the left of the bottom of U-curve. (P326 - Hands on ML)

ii) Learning rate scheduling involves decreasing the learning rate slowly as model learns more (gets closer to convergence). [use Adam optimizer]

iii) Cyclic learning rates

Dynamically change the LR up & down b/w some thresholds & potentially speed-up training.

Other hyperparameter you can tune

1) No. of layers (Deep learning networks)
2) Batch size [How many eg of data your model sees at once.]
3) No. of trees (Decision tree algos)
4) No. of iterations [How many times model goes through the data. (can use early-stopping)]
.. many more ..
Depends on algo used.

Step ④

Analysis / Evaluation:- to estimate how well a model will generalize to out-of-sample data.

a) Evaluation metrics:-

i) classification
- ROC & AUC curve
- Accuracy
- Precision
- Recall
- F1 score
- confusion matrix.

ii) Regression
- MSE (mean squared error)
- MAE (mean absolute error)
- R2 (r-squared)

iii. Task-based metric (create one based on your specific problem).

---

1) **Classification accuracy:-**

Percentage of correct predictions. $\left(\frac{Y\_pred}{Y\_test}\right)$

Null accuracy:-

Accuracy that could be achieved by always predicting the most frequent class.

→ It's a good way to know the min. we should achieve with our models.

→ It gives a baseline to compare our model's accuracy.

∴ classification accuracy is easy, but it doesn't tell you the underlying dist of response class.

→ And it doesn't tell you what type of error your classifier is making.

a) confusion matrix

FP - False Positive
FN - False negative.

	Predicted 0	Predicted 1
Actual 0	TN	FP
Actual 1	FN	TP

TP: which are actually correct

TN: which are actually wrong

FP: they are actually wrong, but predicted as true. ← they actually don't have corona. But we predicte they have.

FN: they are actually true, but predicted as false ← they actually have, but we say they don't

→ Classification accuracy ← $\frac{TP+TN}{TP+TN+FP+FN}$ talks about correct.

How often is the classifier correct?

→ misclassification rate ← $\frac{FP+FN}{TP+TN+FP+FN}$

Recall / sensitivity [True positive rate]

→ How "sensitive" is the classifier to detecting positive instances?

→ when the actual positive is tested, How often it is correct?

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

Out of total positive ones, How many positive I am able to (Recall)

→ It depends on FN, which were actually true, but predicted false.

Actual Pos - TP+FN

Predicted Pos. (TP)

Precision & Recall linked with TP

Precision = Really Pos / Total Pos: actually

$$\frac{TP}{TP+FP}$$

$$\frac{TP}{TP+FN} = \frac{\text{Really Pos}}{\text{Predicted Pos}}$$

→ when a positive value is predicted, How often is the prediction correct?

→ How **"precise"** is the classifier when predicting positive instances.

out of predicted pos, how many really pos.

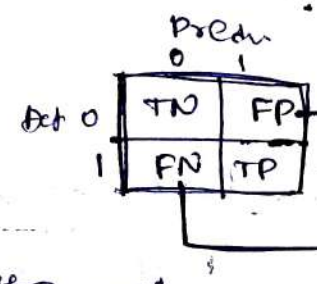
$$\text{Precision} = \frac{TP}{TP+FP}$$

→ It depends on FP, which were actually negative, but predicted true.

Q.) which metrics to focus on?

Eg ①

1 = COVID 19 (+)
0 = − (Healthy)



→ Now analyze the cost

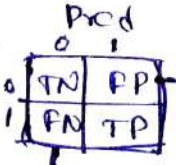
→ we know a sick predicted as healthy could spread a lot more cases.

So, **cost of FN > cost of FP**

Hence it depends on FN, we go for "Recall".

Eg ②

1 = Spam
0 = Not Spam



→ we know if a not spam (imp. mail) goes to spam, that could cause lot of damage.

So, **cost of FP > cost of FN**

Hence, it depends on FP, we go for "precision"

Specificity :-

$$\frac{TN}{TN+FP}$$

False positive rate :-

$$\frac{FP}{TN+FP}$$

FB-Score :-

when you want to consider both FP & FN like we need both sensitivity (recall) & precision, in such cases we choose F-Beta.

General Formula

$$F_\beta = (1+\beta^2) \cdot \frac{\text{Precision} \times \text{recall}}{(\beta^2 \cdot \text{Precision}) + \text{recall}}$$

$$F_\beta = \frac{(1+\beta^2) \times TP}{(1+\beta^2) \times TP + \beta^2 \cdot FN + FP}$$

In beta, there is posibility for 3 ranges.

1) when FP & FN are equally important, like you can't compromise even a bit both are crucial, then we set beta = 1 which is F1-score.

$$F1\text{-Score} = 2 \times \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

2) when both are worse, but FP is drastic than FN. then we go for beta = 0.5 (usually 0-1)

3) when both are worse, but FN is drastic than FP. then we go for beta = 2 (usually 1-10).

---

note

we can also adjust classification threshold to modify performance of classifier.

Roc curve

can see how steadily & sensitivity are affected by various thresholds, without actually changing the threshold.

AUC curve (Area under curve)

If you randomly choose one + & one -, AUC represents the likelihood that your classifier will assign a higher predicted probability to the positive observation.

Analysis of model

1) Feature importance

Which feature contributed most to model
Should some be removed?

2) Training / Inference time

How long does a model take to train?
Is this feasible?

How long does inference take?
Is it suitable for production?

3) Using "what-if" tool

what-if changed thing in data?
How does this effect the outcome?

4) Bias / variance trade-off

→ High bias results in underfitting &
a lack of generalization to new samples

→ High variance results in overfitting due
to the model finding patterns in the
data which is actually random
noise.

Step (5)

Deploying a model

Put the model into production & see how it goes. Evaluation metrics in notebook are great but until it's production, you won't know how it performs for real.

Step (6)

Retrain model

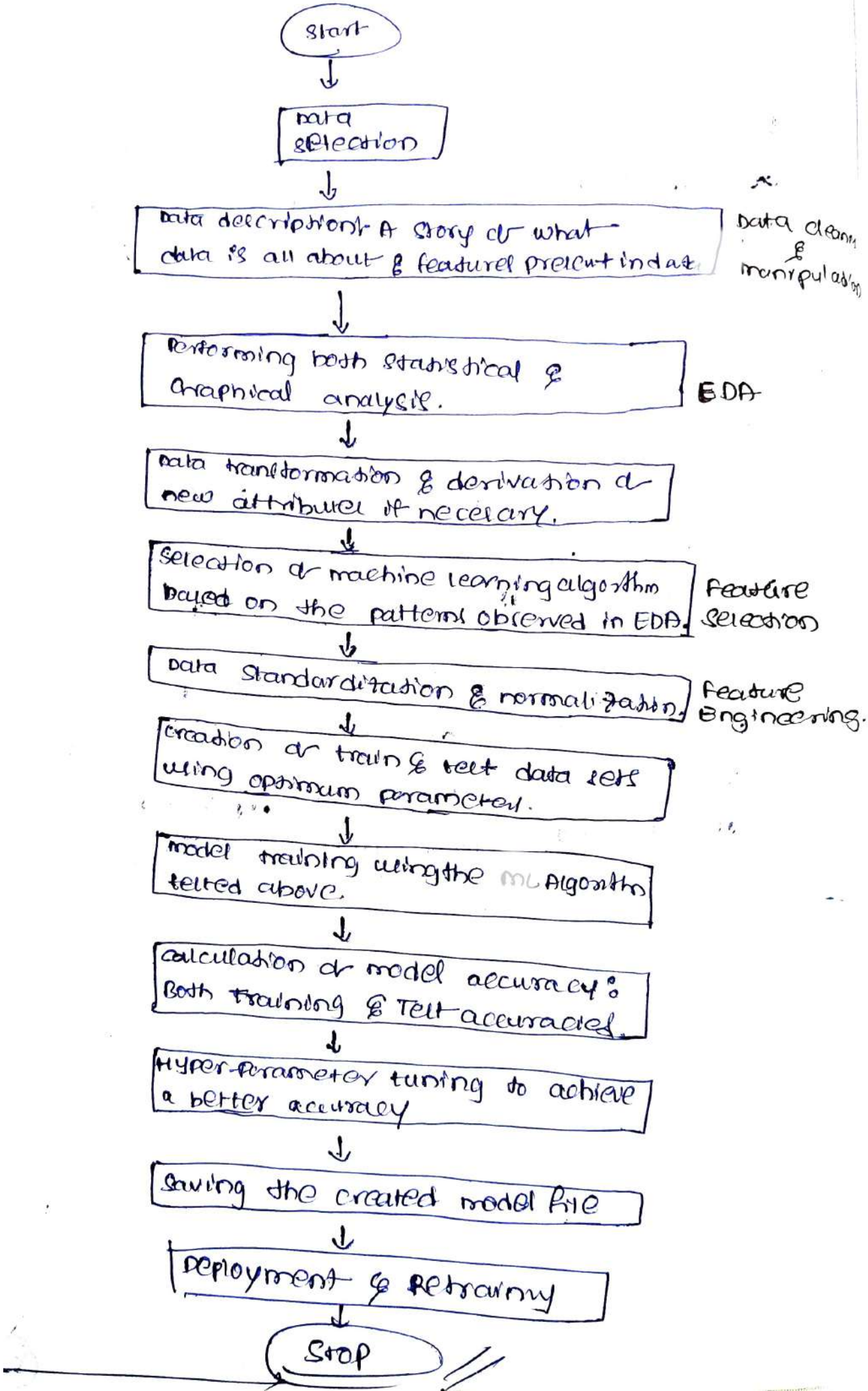
→ See how the model performs after serving based on various evaluation metrics & revisit the above steps as req.

→ You will find your models prediction start to cages or (drift), as in when data sources change or upgrade. this is when you will want to retrain it.

(0,0)

Stop / NO END

So, basically the flowchat is ---



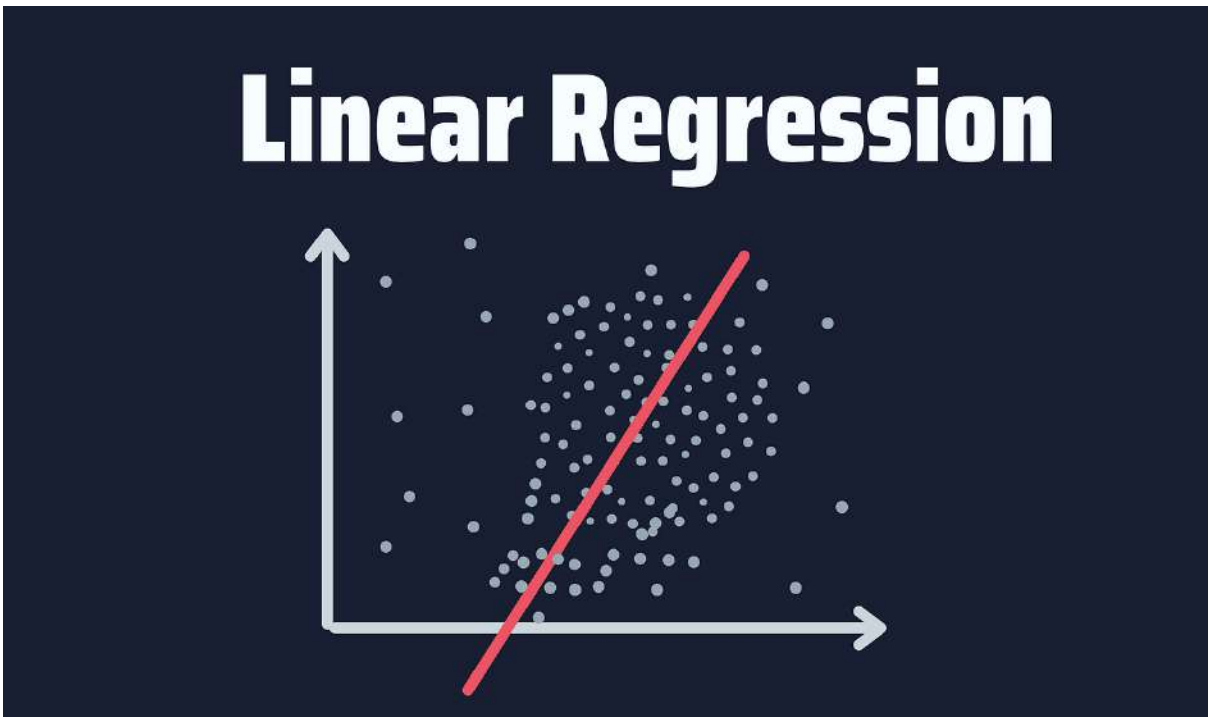


Table of Contents

Understanding the algorithms

① Simple Linear Regression

Linear model in Regression (supervised learning)

Simple linear regression assumes that a linear relationship exists b/w the response variable & the explanatory variable, It models this relationship with a linear surface called a "hyperplane"

→ This will be clear with an eg.
Let's take the height & weight sample of few people

Height	weight
5.6	60
5.7	62
5.8	63
5.9	62
6.1	64
6.2	75

$H \propto W$

$W \propto H$

Now, If I want to know what would be the weight of a person with 6.0 Height?

→ By observation it seems like

$H \propto W$ → to remove proportionality we can multiply a constant

$W = mH$ → But, it may also differ by some value, so, $\pm b$

$W = mH \pm b$ → Now, this is the relation of height & weight.

constants

$w = mH \pm b$ → Now, here we know the value of 'm' & 'b'. then we can predict the weight.

given height

Now, main aim is to find out value of 'm' & 'b'.

→ Let's take 2 samples, (5.6, 60), (5.7, 62),

$60 = m(5.6) + b$ — (1)

$62 = m(5.7) + b$ — (2)

→ By this we get $m = 20$, $b = -52$.

So, our equation $w = 20H - 52$

Let's try with a known height 5.9.

$w = 20(5.9) - 52 = 66$ kg.

Here we got '66', but we are expecting 62.

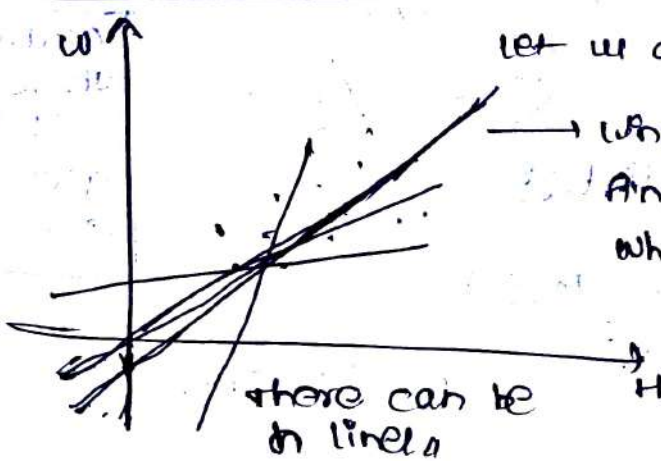
(*) So, may be if we change 'm' & 'b' then we could get a exact value / approx.

Now, Here we get the definition of ML as **establishing a mathematical relationship** of data. So, that it can predict a new data.

→ So, when we say a ML model,

(*) It is nothing but a mathematical relationship which can predict new data.

Plotting 'H' & 'w'



let us assume we have a best fit line.

→ Linear regression always tries to find the linear relation b/w variable which is a straight line.

Hence, $y = mx \pm c$

→ we could find n diff lines.

But that one line which can predict all the values approx. is the "Best fit line"

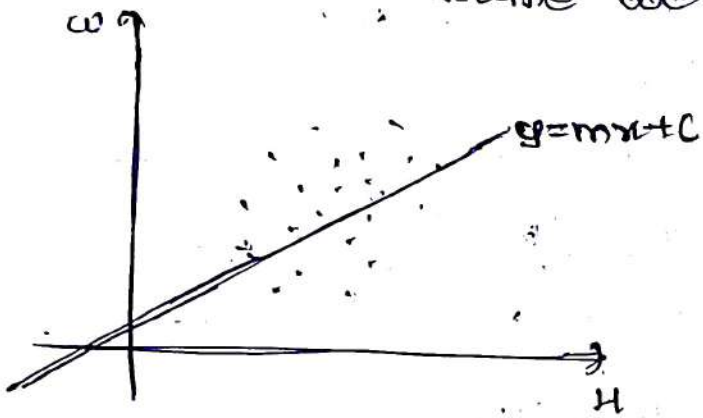
$y = mx + c$ → And this best fit line is called model

→ we have to find the 'm' & 'c'.

→ m-slope, c-Intercept

(*) Evaluating the fitness of the model with a cost function

→ Firstly, let us assume we have a best fit line.



But, now we have doubts.

How did we get this line?
why this line?
what about other lines?

→ Now, if it's the best fit line, then for future if I want the weight of '6.0'.

→ It is nothing but the value of y-coordinate $y = mx + c$ line with x coordinate '6.0'

→ If this line is $W = 20H - 52$ and at $x = 5.9$,

I got $\hat{w} = 66$, but actual $w = 62$.

So, I can see there is a loss / error.

$$\text{loss} = |(y - \hat{y})|$$

(y known, $\hat{y}$ predicted)

Always there will be a slight error

Q.) why did we got this loss?

A may be best of 'm' & 'c' we taken.

Gradient Descent - 48

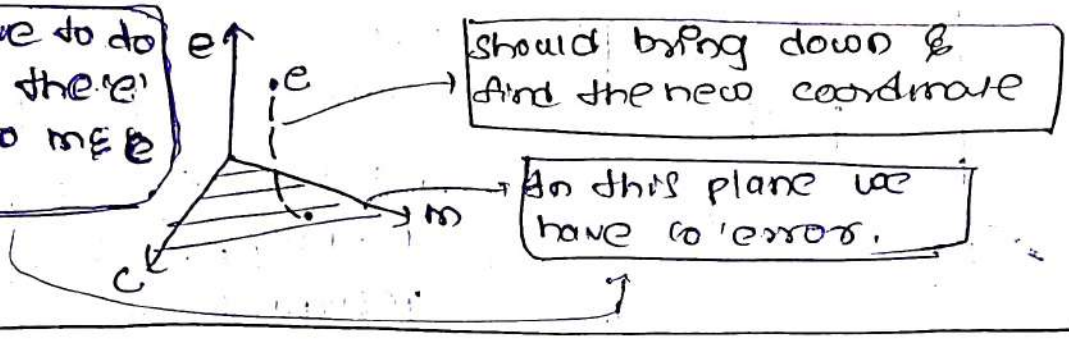
→ Now, to improvize my model strength I have to 'reduce the loss'.

and this loss depends on 'm' & 'c' taken.

→ So, I am supposed to shift m & c such that I end up reducing the loss.

we have loss (e), slope (m), constant (c).

So I have to do is bring the 'e' down to m & c plane



Slowly, by examining each input m & c gets changed and eventually loss decreased. this process is called (learning).

So,

$$m_{new} = m_{old} - \eta \Delta E_m$$

$$c_{new} = c_{old} - \eta \Delta E_c$$

m_{new} - changed one, new

m_{old} - old m value.

η - Learning rate

ΔE_m - Error in m $\left(\frac{de}{dm}\right)$

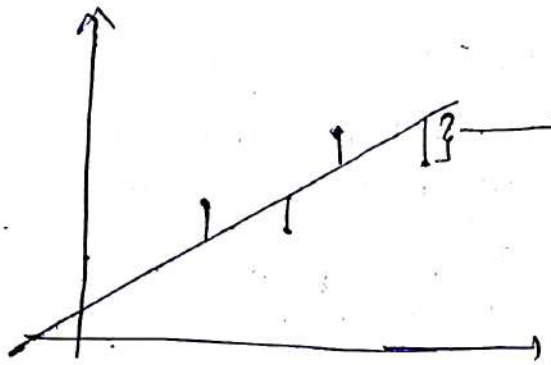
$\eta \rightarrow (0.0001 - 10)$ range.

→ So, here we are taking a new m.

By subtracting a fraction of the error rate from old m that fraction is learning rate.

So, now I am supposed to choose the m & c, which gives a least possible error then it will be the best fit line.

→ A cost function also called a loss func, is used to define & measure the error of a model. the diff. b/w the weights predicted by model & the observed weights in training set are called "residuals" or training error / loss.



→ this is the residual / loss. which is $|y-\hat{y}|$

→ we have to reduce this residual Actually, we have to reduce residual of all data points.

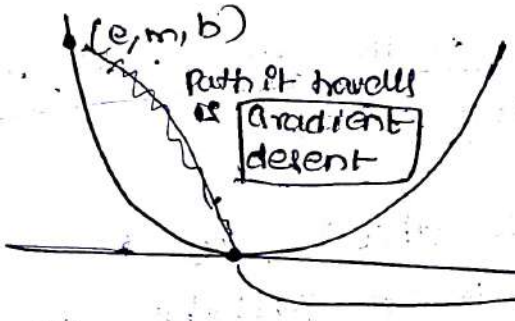
→ we can produce best weight predictor model by minimizing the 'sum of residuals'.

(Sum of residuals) $\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} [y_i - (mx_i + c)]$ → $\hat{y}$

So, to cancel the negative value, we take square

(*) $R(x) = \sum_{i=1}^{n} r_i^2 = \sum_{i=1}^{n} [y_i - (mx_i + c)]^2$ sum of squares of residuals.

Now, it is a quadrant in 3D.



→ So, we are supposed to shift the (e, m, b) to the origin (0, m, b).

where $\frac{de}{dm} = 0$, $\frac{de}{db} = 0$.

As we can see residual is both a func. of m & b, so, differentiating partially w.r.t m & b will give us

$$\frac{\partial E}{\partial m} = \sum_{i=0}^{n} x_i (b + m x_i - y_i)$$

$$\frac{\partial E}{\partial b} = \sum_{i=0}^{n} 2 (b + m x_i - y_i)$$

So, we know for best line, residual should be min. minima of func. occurs where derivative = 0. i.e,

$$\sum_{i=0}^{n} x_i (c + m x_i - y_i) = 0$$

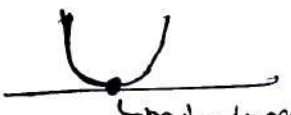
$$\sum_{i=0}^{n} 2 (b + m x_i - y_i) = 0$$

$$\boxed{\begin{aligned} \sum_{i=0}^{n} x_i c + \sum_{i=0}^{n} m x_i^2 + \sum_{i=0}^{n} x_i y_i = 0 \\ \sum_{i=0}^{n} c + \sum_{i=0}^{n} m x_i - \sum_{i=0}^{n} y_i = 0 \end{aligned}}$$

The same eqn. can be written in matrix form as:

$$\begin{bmatrix} \sum_{i=0}^{n} x_i & \sum_{i=0}^{n} x_i^2 \\ n & \sum_{i=0}^{n} x_i \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} \sum_{i=0}^{n} x_i y_i \\ \sum_{i=0}^{n} y_i \end{bmatrix}$$

Ideally, if we'd have an eqn. of one dependent & one independent variable the minima would look like



dependent var $y = mx + c$ independent var

→ the new values for slope & intercept are calculated.

repeat until convergence()

{

$$m_{new} = m_{old} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} (h_0 \cdot x_i - y_i) x_i \right] \quad \leftarrow \frac{dE}{dm}$$

$h_0 = mx$

$$c_{new} = c_{old} - \eta \left[\frac{1}{n} \sum_{i=1}^{n} (h_0 \cdot x_i - y_i) \right] \quad \leftarrow \frac{dE}{dc}$$

}

Accuracy → this is how LR works.

(*) Measuring model strength

RSS - Residual Sum of Squares
TSS - Total sum of Squares

The R^2-squared (R^2) statistic provides a measure of fit.

It takes the (form of proportion) − (proportion explained).

$$R^2 = 1 - \frac{RSS}{TSS}$$

$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad \leftarrow \text{Predicted}$$

because changing

$$TSS = \sum_{i=1}^{n} (\bar{y} - y_i)^2 \quad \leftarrow \text{mean.}$$

If $R^2 = 0.75$ it says that our model fits 75% of dataset.

Adjusted R^2 Statistic

→ As R^2 is just a linear eqn., as we increase the no. of independent variables in our eqn., the R^2 increases as well.

But, that doesn't mean that the new independent variables have any correlation with the output variable.

i.e; R^2 will increase, but is not necessary model yields better results.

→ To rectify this we use adjusted R^2 value which penalises excessive use of such features which don't correlate with the output data.

$$R^2_{adj} = 1 - \frac{(1-R^2)(N-1)}{N-P-1}$$

where

P - no. of predictors

N - total sample size.

If $P=0$, $R^2_{adj} = R^2$

Note:-

→ using training data to learn the values of the parameters for simple linear regression that produce the best fitting model is called ordinary least squares (OLS) (or) linear least square.

Another approach to find m & C.

(To solve OLS)

→ variance is a measure of how far a set of values are spread out.

→ covariance is a measure of how much two variables change together. If the variables increase together, their cov is positive. 0 - no relation, -ve → one increase & one decrease

$$var = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$$

→ variance of explanatory variable.

$$covar = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

→ covariance of the response, explanatory variables.

$$\therefore \beta = \frac{cov(x,y)}{var(x)}$$

$y = \beta x + \alpha$.

$\alpha = \bar{y} - \beta \bar{x}$ → we can get α

Notes

→ The independent variables are uncorrelated with the residual term, also known as "Exogeneity". This in layman term generalises that in no way should the error term be predicted given the value of independent variables.

the error terms have a constant variance.
i.e) Homoscedasticity.

→ the error terms are normaly distributed.

→ No multicollinearity, i.e; no independent variables should be correlated with each other or affect another.

→

model confidence

Q) Is linear regression a low bias / high variance model (or) a high bias / low variance model?

A) It's a "High bias / low variance" model

→ Even after repeated sampling, the best fit line will stay roughly in the same pos. (low variance)

→ But the avg. of the models created after repeated sampling won't do a great job in capturing the perfect relationship. (High bias)

→ Low variance is helpful when we don't have real training data.

```
import statsmodels.formula.api as smf
model_name = smf.ols(formula='y ~ x', data=df).fit()
model_name.conf_int()
```

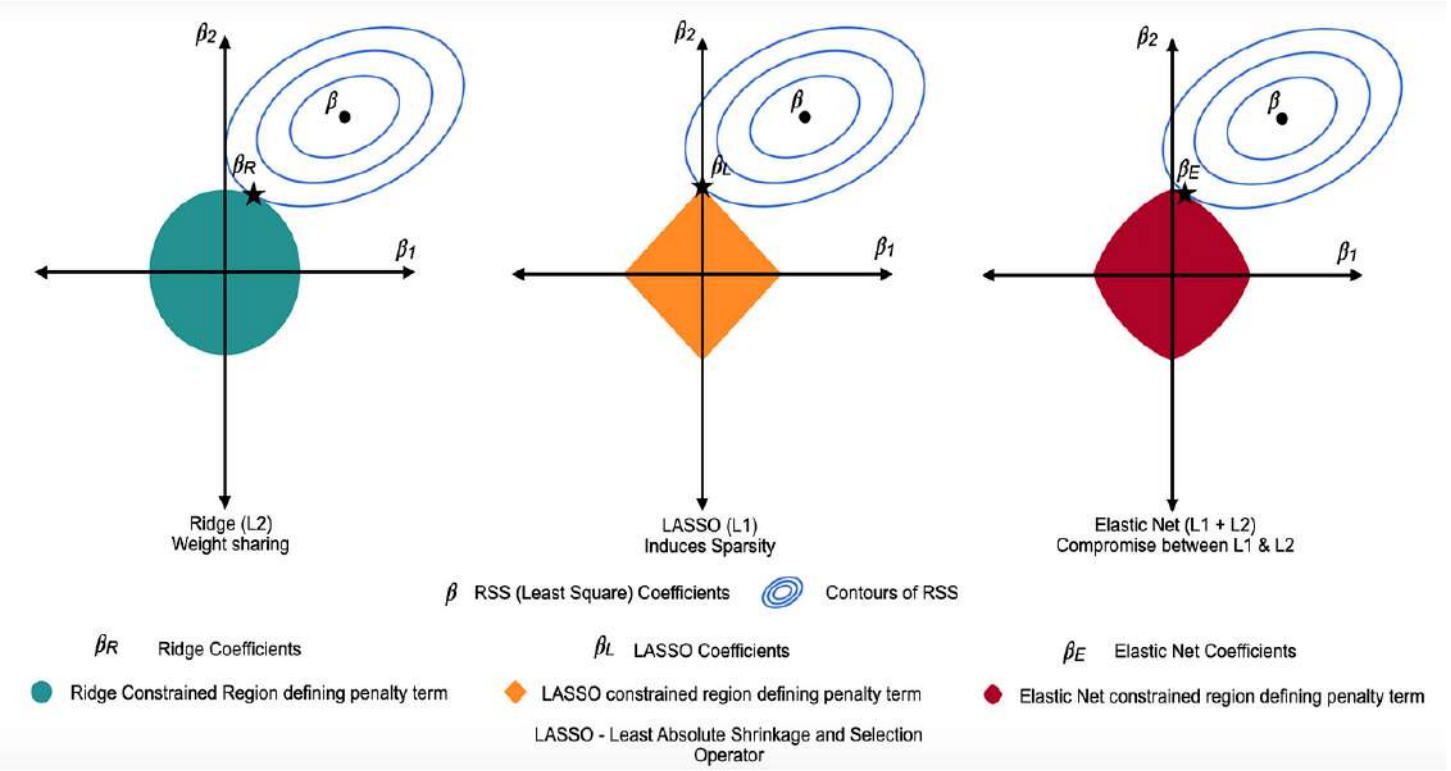


Table Of Contents

(*) Multi collinearity

→ we can define multi-collinearity as the situation where the independent var have strong correlation among themselves.

→ the coeff. in a linear Reg model represent the extent of change in 'y' when a certain $x(x_1, x_2, x_3 \ldots)$ is changed keeping other constant. But if x_1 & x_2 are dependent then this assumption itself is wrong that we are changing one var keeping other constant as the dependent var will also be changed.

It means model becomes a bit flawed.

$y = mx_1 + nx_2 + c$

while,

$x_2 = ax_1 + d$

Independent var (x_2, x_1) are related.

More than one when eqn,

Hence called **multi collinearity** //

→ we have redundancy in our model as two variables (or more than two) are trying to convey the same information.

→ As the extent of collinearity increases, there is a chance that we might produce an "overfitted model".

→ An overfitted model works well with the test data but its accuracy fluctuates when applied to other datasets.

© can result in a dummy variable trap.

→ By the heatmap we can check collinearity.
Generally a correlation greater than 0.9 (or) less than -0.9 are to be avoided.

(#) variance Inflection Factor (VIF) :-

Regression of one X var against other X variables.

$$VIF = \frac{1}{1-R^2}$$

VIF < 5 ✓ perfect

→ If VIF > 5, extreme correlation is avoided.

Remedies for multi-collinearity

(1) Do-nothing :- If corr not extreme / the var not used the ignore

(2) Remove one-var :- like in dummy var trap (onehotencoding)

(3) combine correlated var :- combine variables.

(4) Principle componet Analysis (PCA)

(*) Regularization :-

→ When we use Regr. models to train some data, there is a good chance that the model will overfit the given training dataset.

→ Regularization helps sort this overfitting problem by restricting the degree or freedom of given eq.

Eg)
Simply reducing the no. of degree of a polynomial by reducing their corresponding weights.

Reason for Regularization

→ In a linear eq, we don't want huge coeff, as a small change in coeff can make a large diff. So regular constraints weights or such features to avoid overfitting.

$$y = B_0 + B_1 x_1 + B_2 x_2 + \cdots B_p \cdot x_p$$

$$RSS = \sum_{i=1}^{n} \left(y_i - B_0 - \sum_{j=1}^{p} B_j \cdot x_{ij} \right)^2 \quad (y_i - \hat{y})$$

* To Regularize a model, Shrinkage penalty is added to cost function.

LASSO (Least Absolute Shrinkage & Selection Operator) (L1 form)

→ LASSO regression penalized the model based on the sum of magnitude of the coefficients.

The regularization term is given by, $\text{regularization} = \lambda * \sum |B_j|$

(λ — shrinkage factor)

→ Loss after regularization is:-

New loss func. $RSS + \lambda \cdot \sum_{j=1}^{p} |B_j| = \sum_{i=1}^{n} (y_i - B_0 - \sum_{j=1}^{p} B_j \cdot x_{ij})^2 + \lambda \cdot \sum_{j=1}^{p} |B_j|$

Here, loss is not calculated just by previous values $(y_i - \hat{y})$, now it has been split into several parts which decreases loss.

Ridge Regression (L2 form)

→ Ridge Regression penalized the model based on the sum of squared of mag of coeff.

$\text{regularization} = \lambda * \sum |B_i^2|$

λ — can be calculated by cross validation

→ loss after regularization,

$$RSS + \lambda \cdot \sum_{j=1}^{p} B_i^2 = \sum_{i=1}^{n} (y_i - B_0 - \sum_{j=1}^{p} B_i \cdot x_{ij}) + \lambda \cdot \sum_{j=1}^{p} B_i^2$$

Note:- consider B_1 & B_2 be coeff of LR & $\lambda = 1$.

(1) For LASSO, $|B_1| + |B_2| \leq S$

(2) for Ridge, $B_1^2 + B_2^2 \leq S$

→ where 'S' is the max value the eqn can achieve. At

Elastic net

→ middle ground b/w Ridge & Lasso.

→ the regularization term is a simple mix of both Ridge & Lasso, and you can control mix ratio α.

$$\text{Lenet}(\vec{\beta}) = \frac{\sum_{i=1}^{n}(y_i - x_i\vec{\beta})^2}{2n} + \lambda\left[\left(\frac{1-\alpha}{2}\right)\cdot\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\beta_j|\right]$$

→ when 'α' is the mixing parameter.

Ridge ($\alpha=0$)

Lasso ($\alpha=1$) //

(*) difference between Ridge & LASSO

→ Ridge Regression shrinks the coeff for those predictors which contribute very less in the model but have huge weights, very close to 0.

$y = 0.04x + 1e4$

$R^2 = 0.0016$ — very less

thus final model will still contain all those predictors, though with less weights.

→ where as in LASSO, the L1 penalty does reduce some coeff exactly to zero, when we use a sufficiently large dummy parameter λ.

So, in addition to regularizing, LASSO also performs feature selection.

Q- Why use Regularization?

A → It helps to reduce the variance of the model, without a substantial increase in the bias.

→ If there's variance in the model that means the model won't fit well for dataset other than training data (which is called overfitting).

→ the tuning parameter (λ) controls this bias & variance trade off. selected using cross-validation

→ when (λ) is increased to certain limit it reduces the variance without losing any important data.

→ But after certain limit model will start losing important prop which will increase (bias) in the data.

Q) what should be used plain linear, Ridge, LASSO or Elastic net?

A)

→ It is always preferable to have atleast a little bit of Regularization, so generally avoid plain linear Reg.

→ Ridge is a good default,

→ But if you suspect that only a few features are actually useful, you should prefer LASSO/EN, since they tend to reduce the useless features weights down to zero.

→ In general Elastic net is preferred over LASSO since LASSO may behave erratically when the no. of features is greater than no. of training instances or when several features are strongly correlated.

whenever, mean is not 0.
it's better to do StandardScaler

Note:-

① StandardScaler() → $\frac{x-\bar{x}}{SD}$ $\bar{x}=0$ $SD=1$

② Two ways to find multicollinearity → correlation & VIF

③ 4 ways to remove multicollinearity.

④ train set, high accuracy; test set, less accuracy.
i.e. overfitting, so regularize it.

→ Even after checking Lasso and Ridge, if it gives the same accuracy values, then you can't is not over-fitted.

i.e. just by having huge difference in train accuracy & test accuracy than it doesn't just mean overfitted. test before judge. on this case model is not problem, more data is needed.

⑤ The linear in Linear Regression does not talk about the degree or polynomial eq in terms of dependent variables.

instead, it talks about the degree of coeff.

$y = a + bx + cx^2 + \cdots nx^n$

it's not talking about power of x, but the power of a, b, c etc. And their powers 1.

Hence it is linear Regression.

Polynomial Regression

→ It is a mechanism to predict a dependent var based on the polynomial relationship with the independent variable.

In the eq, $y = a + bx + cx^2 + \dots nx^n + \dots$

max. power of x is called degree of polynomial.

∴ Deg ① → $y = a + bx$

Deg ② → $y = a + bx + cx^2$

When to use?

⊛ If data is scattered in a polynomial curve shape (not linear) then keep trying different degree until the line fit to data.

Codes

```
from sklearn.preprocessing import PolynomialFeatures
Poly_reg = PolynomialFeatures(degree=2)
X_Poly = Poly_reg.fit_transform(X)
```

→ Keep changing degree until it fit to data.

✱ Logistic regression (it not Regression model)

⑧ Logistic Regression

Linear model for classification (supervised learning)

✱ (Probability based model)

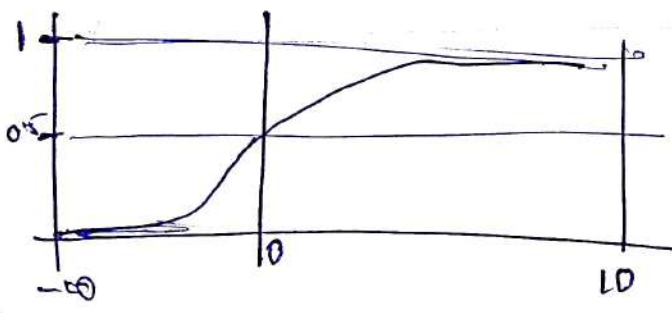
→ Logistic Regression is used for performing "classification problems".

→ It calculates the probability that a given value belongs to a specific class. If the probability is more than 50%, it assigns the value to that particular class,
else if probability is less than 50%, the value is assigned to the other class.

∴ we can say that logistic Regr. acts as a binary classifier.

✱ Here, As we give input x, we get y, where y is the probability or given var x belonging to a certain class.
thus, it is obvious that the value of y should we between 0 & 1.

So, we use Sigmoid function as the underlying function in Logistic Regression.



$$\sigma(x) = y = \frac{1}{1+e^{-x}}$$

Q. Why we sigmoid?

① the sigmoid function's range is bounded between 0 & 1, which is what we need.

② Easy to calculate than other functions which is useful during gradient descent calculation.

③ It is a simple way to introduce non-linearity to model

→

the logistic func is given as,

$$P(x) = \frac{e^{B_0 + B_1 x}}{1 + e^{B_0 + B_1 x}}$$

we have $B_0 + B_1 x$

$$P(x) = \frac{1}{1 + e^{-(B_0 + B_1 x)}} = \frac{e^{B_0 + B_1 x}}{1 + e^{B_0 + B_1 x}} \quad -①$$

$$1 - P(x) = 1 - \frac{e^{h(\theta)}}{1 + e^{h(\theta)}} \quad (h(\theta) = B_0 + B_1 x)$$

$$1 - P(x) = \frac{1}{1 + e^{h(\theta)}} \quad -②$$

Now Divide ①, ②

$$\frac{①}{②} \Rightarrow \frac{P(x)}{1 - P(x)} = e^{h(\theta)} = e^{B_0 + B_1 x}$$

$$B_0 + B_1 x = \log\left(\frac{P(x)}{1 - P(x)}\right)$$ → logarithmic function.

Now we have,

$$\log\left(\frac{p(x)}{1-p(x)}\right) = \beta_0 + \beta_1 x$$

(logit function)

If we input (x), we will get $p(x)$ (probability)

→ And we get $p(x)$, based on the line with β_0, β_1 constants.

This is how logistic regression works

Here, if we have two classes $(0, 1)$

$$y = \begin{cases} 0 & , \text{if } p(x) < 0.5 \\ 1 & , \text{if } p(x) \geq 0.05 \end{cases}$$

0.5 – threshold

we can change threshold

→ Based on the scenario, we should change threshold.

Cost function / loss function

For a single training instance,

$$\text{cost}\left(p(x), y\right) = \begin{cases} -\log(p(x)) & , \text{if } y = 1 \\ -\log(1-p(x)) & , \text{if } y = 0 \end{cases}$$

If $p(x) = 1$ & $y = 1$, cost / loss $= 0$

If $p(x) = 0$ & $y = 0$, cost / loss $= 0$

But,

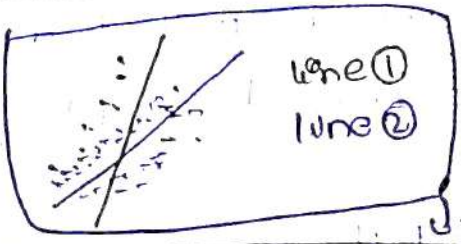
If $p(x) = 0$ & $y = 1$, cost / loss $\to \infty$

If $p(x) = 1$ & $y = 0$, cost / loss $\to \infty$

→ cost func for whole training set is given as

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} \left[y_i \cdot \log(\hat{p}_i) + (1-y_i) \cdot \log(1-\hat{p}_i) \right]$$

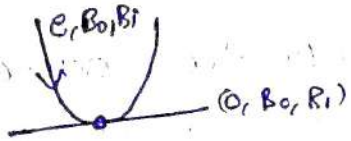
where, m - rows
y_i - expected y
$\hat{p}_i$ - predicted y



→ In logit func, RHS we have a line, And we will keep on adjusting the line w.r.t error so that we'll get correct probability // for from data

→ the values of parameter (θ) for which the cost func is (min) is calculated using the gradient decent algorithm (as showed in LR).
the partial derivative for above cost func is,

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[\sigma(\theta^T \cdot x_i) - y_i \right] \cdot x_j$$



multinomial logistic Regression

→ we can extend Logistic Regression for multi-class classification. the logic is, we train our model for each class and calculate the probability that a specific feature belongs to that class. once we have trained the model for all the classes, we predict a new values class by choosing that class for which prob (class) is maximum.

* we rarely use it this way, because we have many other classification models for these scenarios.