LECTURE - 48

We know that,

$\max \Sigma y_{act} * y_{pred}$ where

$y_{act} * y_{pred}$ is sum of signed distances

$$\Rightarrow w^*, w_0^* = \arg\max_{w, w_0} \left\{ \sum_{i=1}^{n} y_{i\,act} (w^T \cdot x_i + w_0) \right\}$$

From the above eq^n we have a outlier problem.

So, let us consider function 'f' for

$y_{act} * y_{pred}$

$$\Rightarrow w^* = \arg\max_{w} \{ \Sigma y_{act} * y_{pred} \}$$

$$\Rightarrow w^* = \arg\max_{w} \{ \Sigma y_{act} * w^T \cdot x_i \}$$

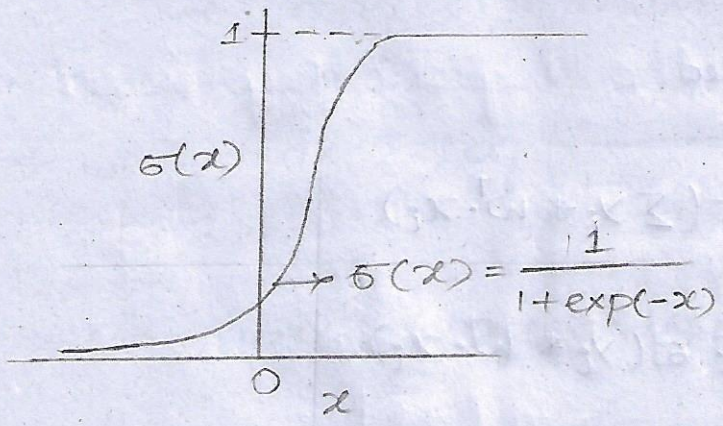
$$\Rightarrow \boxed{w^* = \arg\max_{w} [q(f)]}$$

where 'q' is the SIGMA FUNCTION

$\Rightarrow q(f)$ is also called as LOGISTIC FUNCTION.

↳ What ever we provide to 'g' its output turns to lie between '0' and '1'.

↳ Instead of sigma function, if we use square of the function the distance increases.



↳ the sigmoid function is also called as S-shaped curve.

$$\Rightarrow \boxed{\sigma(x) = \frac{1}{1+\exp(-x)} \quad (or) \quad \frac{1}{1+e^{-x}}}$$

where $\sigma(x)$ is the sigmoid function representation.

NOTE:

→ If the values are large, it tapers up and tends to rest them at '1'.

Now, we can write the sigmoid function as

$$\sigma(f) = \frac{1}{1+e^{-f}} \quad \text{where } f = y_{act} * w^T \cdot x_i$$

↳ Lets take the sigmoid function on top of

$$\max \Sigma\, y_{act} * y_{pred}$$

$$\Rightarrow w^* = \arg_w \max \sigma(\Sigma y_i * w^T \cdot x_i)$$

$$\Rightarrow w^* = \arg_w \max \Sigma\, \sigma(y_i * w^T \cdot x_i)$$

$$\Rightarrow \boxed{w^* = \arg_w \max \Sigma \frac{1}{1+e^{-(y_i * w^T \cdot x_i)}}}$$

The above eqⁿ is the optimization eqⁿ.

Over here,

we changed max. sum of signed distances

i.e., $\max \Sigma (y_i * w^T \cdot x_i)$ ↳ (A) to max. sum of sigmoid distances i.e., $\max \Sigma \frac{1}{1+e^{-(y_i * w^T \cdot x_i)}}$ → (B)

where A → Problem is outlier

B → less impacted because of outlier.

$$\Rightarrow \boxed{w^* = \arg\max_{w} \cdot \sum \frac{1}{1+\exp\{-y_i \cdot w^T \cdot x_i\}}}$$

$$\left[\because e^{+x} = \exp\{x\}\right]$$

So, According to the optimisation theory,

→ $\max(f)$ follows $\max \cdot \log(f)$

$$\Rightarrow \max(f) = \max\{\log(f)\}$$

$$\Rightarrow w^* = \max \cdot \sum \log\left(\frac{1}{1+\exp\{-y_i \cdot w^T \cdot x_i\}}\right)$$

$$\Rightarrow w^* = \max \cdot \sum \left\{\log(1) - \log(1+\exp\{-y_i \cdot w^T \cdot x_i\})\right\}$$

$$\left[\because \log\left(\frac{a}{b}\right) = \log a - \log b\right]$$

As we know that $\log 1$ is zero

$$\Rightarrow \boxed{w^* = -\max \sum \{\log(1+\exp\{-y_i \cdot w^T \cdot x_i\})\}}$$

Optimisation theory also says that,

→ $\max(-f)$ follows $\min(f)$

$$\Rightarrow \max(-f) = \min(f)$$

From the above eq^n we can say that

$$\Rightarrow \boxed{w^* = \arg\min_{w} \sum_{i=1}^{n} \log\left(1 + \exp\{-y_i * w^T \cdot x_i\}\right)}$$

This i.e., the above equation is called as the "LOGISTIC REGRESSION OPTIMISATION EQUATION".

Now lets make the logistic Regression in simple manner.

STEP-1: Randomly pick w_{old}

STEP-2: $w_{new} = w_{old} - \eta \left[\frac{\partial f}{\partial w}\right]_{w_{old}}$

STEP-3: Repeat the Step-2 until it converges.

So, the Task is

TASK: Find a line that best seperates +ve's from -ve's

where line - $f(m, c)$

best seperates - max. sum of signed distances.

$$\Rightarrow w^* = \max \sum y_i * w^T \cdot x_i$$

$$\Rightarrow w^* = \max \cdot \sum \frac{1}{1+\exp\{-y_i * w^T \cdot x_i\}}$$

$$\Rightarrow w^* = \max \cdot \sum \log\left[\frac{1}{1+\exp\{-y_i * w^T \cdot x_i\}}\right]$$

$$\Rightarrow w^* = \max \cdot \sum -\log(1+\exp\{-y_i * w^T \cdot x_i\})$$

↓ It is because of sigmoid fn.

$$\Rightarrow w^* = \arg\min_w \sum \log(1+\exp\{-y_i * w^T \cdot x_i\})$$

From the above eqn, we can also write as

$$\Rightarrow \max \sum y_i * w^T \cdot x_i = \min \sum \log(1+\exp\{-y_i * w^T \cdot x_i\})$$

↳ As $\log 1$ is constant & its value is '0',

It can be written in the form of

$$\log[\exp\{x\}] = x.$$

$$\Rightarrow \boxed{\min \cdot \sum -y_i * w^T \cdot x_i = \max \sum y_i * w^T \cdot x}$$

It is because, $\max(-f) = \min(f)$ &
$\min(-f) = \max \cdot (f)$

LOGISTIC REGRESSION ASSUMPTIONS:

→ Binary Logistic Regression requires the dependent variable to be binary.

→ For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome.

→ Only the meaningful variables should be included.

→ The independent variables should be independent of each other i.e., the model should have little or no multicollinearity.

→ The independent variables are linearly related to the log odds.

→ Logistic Regression requires quite large sample sizes.

KNN ALGORITHM:

K-NEAREST NEIGHBOR algorithm is one of the simplest classification algorithm and it is one of the most used learning algorithm.

→ K-NN is a non-parametric, lazy learning algorithm.

→ Its purpose is to use a database in which the data points are seperated into several classes to predict the classification of a new sample point.

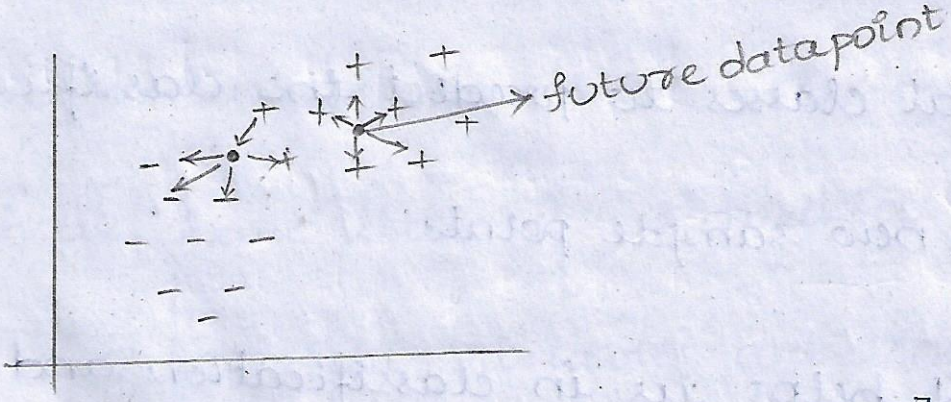
↳ K-NN helps us in classification and also we can convert into regression.

HOW K-NN IS CALCULATED?

→ Determine parameter K = number of nearest neighbors.

→ Calculate the distance between the query point/instance and all the training samples.

→ Sort the distance and determine the nearest neighbors based on the K-th minimum distance.



⇒ For +ve's ⇒ K = 5 − [+, +, +, +, +]

In this case → +ve's = 5, −ve's = 0

⇒ For −ve's ⇒ K = 5 − [+, +, −, −, −]

In this case → +ve's = 2, −ve's = 3

TASK - Given any query point (x_q) we want to find its class by looking at is k-nearest neighbors.

k-nearest neighbors - The Geometrical distance i.e., Euclidean distance.

WHERE IS K-NN USED?

↳ As we know that K-NN is a simple algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions)

→ KNN is been used in statistical estimation and pattern recognition.

↳ the main advantage of K-NN over other algorithms is that KNN can be used for multiclass classification.

WORKING OF K-NN ALGORITHM:

↳ This algorithm uses 'FEATURE SIMILARITY' to predict the values of new data points which further means that the new data point will be assigned a value based on how closely it matches the points in the training set.

We can understand its working with the following steps:

STEP-1: For implementing any algorithm, we need a dataset. So, during the first step of K-NN we must load the training as well as the test data.

STEP-2: Next, we need to choose the value of K, i.e., the nearest data points.

'K' can be any integer.

STEP-3: For each point in the test data, we need to do the following:

- Calculate the distance between the test data and each row of training data with the help of any of the method namely: EUCLIDEAN, MANHATTAN (or) HAMMING DISTANCE.

  The most commonly used method to calculate distance is EUCLIDEAN.

- Now, based on the distance value, sort them in ascending order.
- Next, it will choose the top K rows from the sorted array.
- Now, it will assign to the test point based on most frequent class of these rows.

STEP-4: End.

For Example:

we have some values as

X	y
1	150
2	5
3	0.1
⋮	⋮
1000	0.5

→ [0.1, 0.2, 0.5 ...., 150]

[3, 150, 1000]

+ve +ve +ve

+ve's = 3 & -ve's = 0

⇒ x_q = +ve

→ Given - d_{train} (Training data), K, x_q

STEP-1: Initialize distance.

STEP-2: For each data point in d_{train}, compute the distance of x_q from the data point choosen.

↳ APPEND (DISTANCE, INDEX OF DATAPOINT)

⇒ O/p - [(150,1), (5,2), (0.1,3) ...., (0.5,1000)]

↓

Tuples inside the list = list of the distances

STEP-3: Sort the distance list according to the distance.

⇒ O/p - [(0.1, 3), (0.3, 150), (0.5, 1000 ..., (150, 1)]

where → 0.1 - distance of x_q from the data point in d_{train}

→ 3 - index of d_{train}

STEP-4: Pick the 1st K-tuples from the distance list.

⇒ nearest_neighbors = distance [:3]

↓

by slicing.

STEP-5: Counting the class i.e.,

(+) → pos_count = 0

(-) → neg_count = 0

for neighbour in nearest_neighbours:

check for neighbour class.

if class is +ve → pos_count = 1

else class is -ve → neg_count = -1

STEP-6 : if (pos_count > neg_count):

output - pos_count

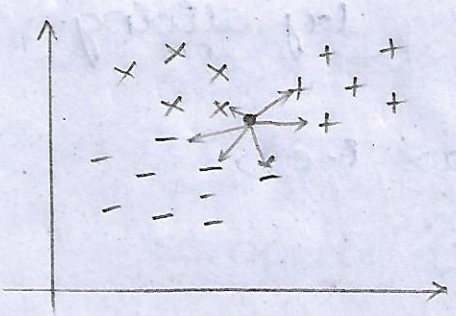
else:

output - neg_count

HOW TO FIND NEAREST POINTS IN K-NN?

↳ It starts by calculating the distance of point x from all the points.

↳ then it finds the 3 nearest points with least distance to the point x.



→ For +ve's ⇒ K = 6

- [+, +, −, −, −, ×]

+ve's = 2, −ve's = 3, x's = 1.

In a binary classification, 'K' must be ~~TRUE~~ 'ODD' and then it holds 'TRUE'.

WHY 'K' SHOULD HOLD ODD VALUES?

'K' should be odd so that there are no ties in the voting.

→ If square root of a number of data points is even, then add or subtract 1 to it, to make it odd.

* → It is advisible to take odd values for binary classification to avoid the ties i.e., two classes labels achieving the same score.

WHAT HAPPENS WHEN YOU INCREASE 'K' VALUE?

If we increase K, the areas predicting each class will be more "smoothed" since its the majority of the K-NN which decide the class of any point.

→ So, If K = 1, then the object is simply assigned to the class of that single nearest neighbor.

HYPERPARAMETER TUNING:

Hyperparameter tuning (or) optimization is the problem of choosing a set of optimal hyperparameters for a learning algorithm.

↳ Its a parameter whose values are used to control the learning process.

↳ By contrast, the values of other parame-ters (typically node weights) are learned.

↳ The ultimate goal is to find an optimal combination of hyperparameters that minimizes a predefined loss function to give better results.

K = > class ≥ multiple of class

NOTE:

Use an even number for k when you have an odd number of classes i.e., the inverse of the even number of classes.

FIT FUNCTIONS USED:

In logistic Regression - $\min \Sigma$ signed distance

In linear Regression - $\min \Sigma e^2$

In KNN classification - Memorizing the data.

↳For the evaluation process we us accuracy.

ACCURACY:

It is the fraction of predictions of our model doing right.

→ Formally, accuracy is defined as the Number of correct predictions to its total number of predictions.

↳ Accuracy is one metric for evaluating classification models.