(**) Linear Regression → useful while predicting a continuous spectrum of output.

↓

Now, given an X, we want the algo to predict; the corresponding Y

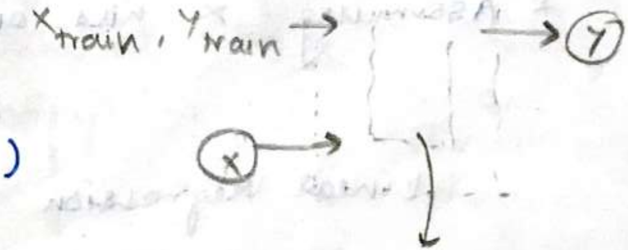
↓

X_{train} and Y_{train} will be used by the machine, to come up (mapping) with a function (best possible) so that future prediction or test predictions can be as acurate as possible

(our main idea)

↓

fitting



a function b/w X and Y, using which it will come up with values for future case of X.
(mapping)

* best fit line

* Function / Hypothesis

that this function fits our data / problem

(Linear regression) tries to put a linear function out there. i.e it assumes (linear relation) b/w x and y

$$\boxed{y = m_1 x_1 + m_2 x_2 + m_3 x_3 + b}$$

output

dependent on each feature linearly

m_n showcases the dependency of y on the n^{th} feature

this dependency can be found out by varying m_1, m_2 and m_3.

different features

The dependency of y on different features can be maintained using the values of m_1, m_2 and m_3

The 4 parameters m_1, m_2, m_3 and b are to be found out to reach y from x that is what the training data is for. (main objective is to reach the eqn. for that we need the 4 parameters)

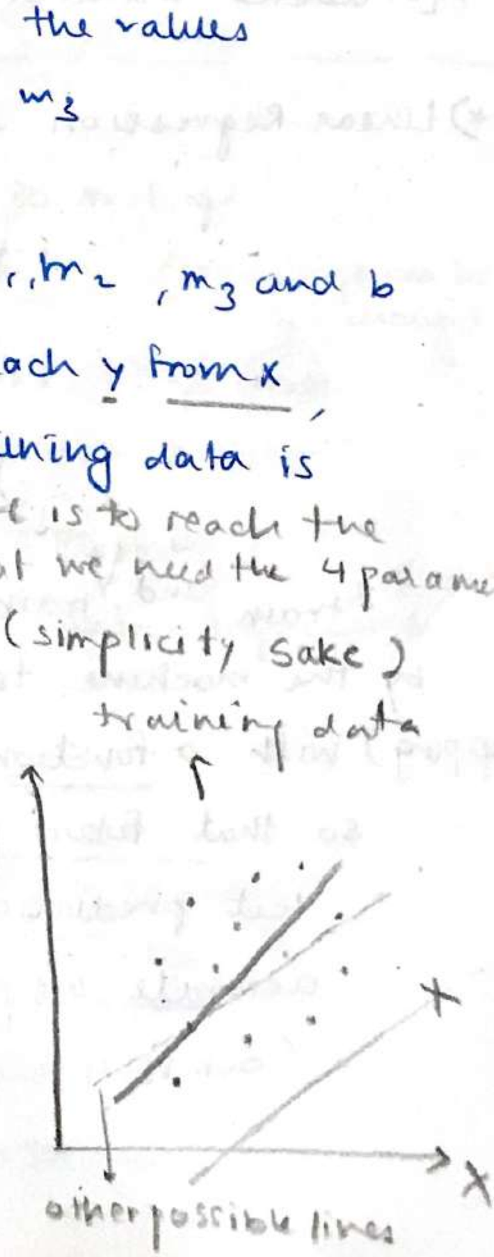
* Assuming x has only one feature (simplicity sake)

x ⟷ y

Linear Regression will try and fit it with one line, such that it is near to all points i.e. combined (error) is (minimum)

how far we are from the line

line selected to minimize error

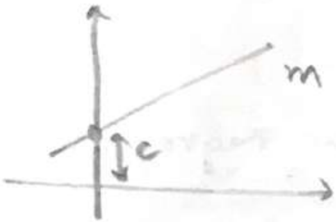


* To find the best fit line, we need to find m and c

(for 1 feature)

for n features : n+1 parameters (including

$m_1, m_2, \ldots, m_n, c$)



$y = mx + c$ → intercept

slope

* given the best fit line is found for training set,

we will just put it in the eqn. and find the

Y_{pred} for X_{test}.

i.e. $\boxed{y' = mx' + c}$

(found using training data)

(6 a)

* Coefficient of determination

↓

To see how well the algo. is performing, we were just drawing a line and calculating how far the datapoints are from the line

↓

(not the best approach)

↓

n features → n variables

↓

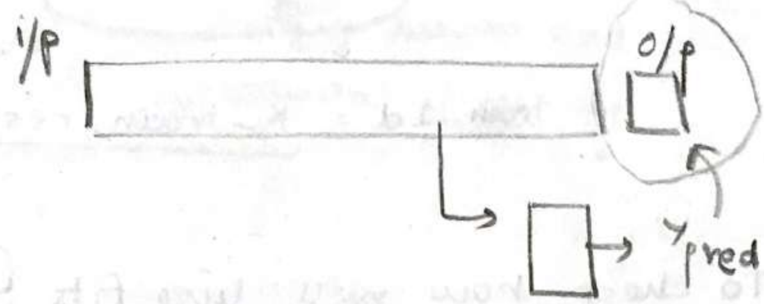
n dimensions

↓

(impossible to plot) ← complexity of graphs ↑

* objective way to find performance of algo.

way to compare the prediction and output and come up with a score



↓

i) important to compare different algorithms

ii) for n feature dataset

2	3	4	Truth (actual outputs)
1.8	2.6	3.5	Prediction

Scoring → Coefficient of Determination

$$\boxed{1 - \frac{u}{v}}$$

$$\boxed{u = \sum (y_i^T - y_i^P)^2}$$

$$v = \sum (y_i^T - y^T_{mean})^2$$

$$COD = 1 - \frac{u}{v}$$

$$= 1 - \frac{\sum (y_i^T - y_i^P)^2}{\sum (y_i^T - y^T_{mean})^2}$$

* Higher the score, better the algorithm. (b/w 0 and 1 preferably)

$u \rightarrow$ sum of (errors squared)

$v \rightarrow$ assuming the ~~error~~ prediction is mean for each datapoint, then sum of error squared.

i.e. we assume mean can be the worst value, we can predict.

for each $x_i \rightarrow y^T_{mean}$ → if predicted value $= y^T_{mean}$

and if all $y_i^T = y^T_{mean}$, then $COD = 1 - \frac{e}{e}$

$= 1 - 1 = 0.$

LOL. :D

* if we make predictions worse than mean, then COD can be -ve

* if we predict all true values, then $COD = 1 - \frac{0}{e}$

numerator = 0

$= 1$

(upperband)

* already implemented in sci-kit learn by

the name of (score)

↓

alg1.score(X_test, Y_test)

↓

takes out Y-prediction and compares with y-test to give us the score

alg1. score (X-train, Y-train)

*) The score value is Judged based on the data and problem at hand. In some cases 0.7 might be a great score and in some cases 0.9 may also not be enough.

* algorithm may perform better on testing data, when compared with training data.

* random split may split differently in the next case

↓

(so the scores may also vary.)

↓

* Mathematical model behind linear regression

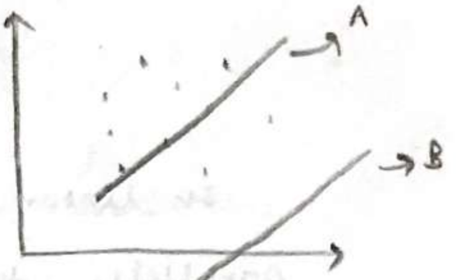
on how it finds the bestsuited line

↓

an objective approach to find the best

↓

Error function



x_i, y_i ← ith datapoint

↓

input vector

↓

$[x_1^i, x_2^i, x_3^i, \dots x_n^i]$

↓

n features
related variables

$[x_1^i]$ → feature

↓

ith data point

Error function value should be very high for B as compared to A

↓

the line predicted → $m_1x_1^i + m_2x_2^i + m_3x_3^i + \dots + m_nx_n^i + c$

↓

feature

i → not exponent, but ith training example

↓

Error (for certain training point) : $y_i - (m_1x_1^i + m_2x_2^i + \dots + c)$

↓ Summation over entire dataset

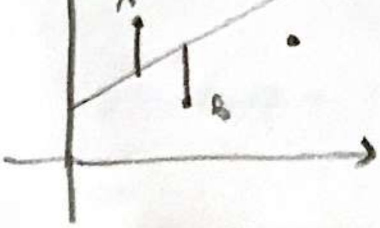
Error for whole training set : $\sum_i^N (y_i - (m_1x_1^i + m_2x_2^i + \dots c))$

↓

In this case, there will be a problem that A and B will have diff. signs, hence they will cancel each other out

↓

(not desirable).

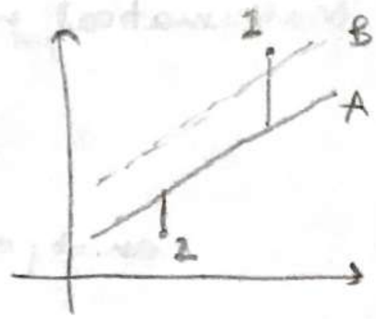


(66)

$$\sum_{i}^{N} (y_i - (mx_i + c))$$

↓

$$\sum_{i}^{N} |y_i - (mx_i + c)|$$



↓

In linear case, when we move 2 lines parallely, then the sum of errors will remain the same, as sum of distances remains the same (example point 1 and 2 from A and B)

↓

$$\sum_{i} (y_i - (mx_i + c))^{2}$$ → for 1D.

error function / cost function

In this case for lines A and B, A will be suited more, since for B, the distance from point II is high.

A is more suited.

↓

We are punishing more for more distance by squaring

for n dimension,

$$\sum_{i} (y_i - (m_1 x_1^i + m_2 x_2^i + m_3 x_3^i + \ldots + m_n x_n^i + c))^2$$

x_1^i ↓ feature for ith data point

* current calculations only for 1 feature

↓

[come up with an intuition for n dimensions later]

$$cost = \sum_i (y_i - (mx_i + c))^2$$

↓

we need to find the line for which the cost function is minimum.

↓

we need m and c for that line

↓

find suitable m and c, so that cost is minimized.

$$cost(m, c) = \sum_i (y_i - (mx_i + c))^2$$

$$\frac{\partial(cost)}{\partial m} = 0 \quad \Bigg| \quad \frac{\partial(cost)}{\partial c} = 0.$$

↓

2 variables, 2 eqns

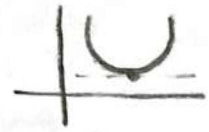
↓

m and c.

** → quadratic

↓

parabolic



↓

corresponding m and c for cost function to be minimum

* $cost = \sum_i (y_i - (mx_i + c))^2$

$$\frac{\partial(cost)}{\partial m} = \sum_i 2(y_i - (mx_i + c))(-x_i) = 0$$

$$= \sum_i y_i x_i - m\sum_i x_i^2 - c\sum_i x_i = 0$$

$$= \sum_i \frac{y_i x_i}{N} - m\sum_i \frac{x_i^2}{N} - c\sum_i \frac{x_i}{N} = 0 \quad (1)$$

$\frac{\sum x_i}{N}$ = mean of x = x.mean()

68

$$\frac{\sum x_i^2}{N} = (x^* x).\text{mean}()$$

to get to mean

$$\frac{\sum x_i y_i}{N} = (x^* y).\text{mean}()$$

$\times$ $$\text{cost} = \sum_i (y_i - (mx_i + c))^2$$

$$\frac{\partial \text{cost}}{\partial c} = \sum_i 2(y_i - (mx_i + c)) = 0$$

$$\Rightarrow \sum_i y_i - m\sum_i x_i - nc = 0$$

for complete set

same across all inputs

$$\Rightarrow \sum_i y_i - m\sum_i x_i = nc$$

$$\Rightarrow \sum_i \frac{y_i}{N} - m\sum_i \frac{x_i}{N} = c$$

$$c = y.\text{mean}() - m^* x.\text{mean}()$$

$$\frac{\partial \text{cost}}{\partial m} = x^* y.\text{mean}() - m(x^* x).\text{mean}() - c^* x.\text{mean}()$$

$$\Rightarrow x^* y.\text{mean}() - m(x^* x.\text{mean}()) - c^* x.\text{mean}() = 0$$

$$\Rightarrow x^* y.\text{mean}() - m(x^* x.\text{mean}()) - (y.\text{mean}() - m^* x.\text{mean}())^* x.\text{mean}() = 0$$

$$\Rightarrow (x \cdot y).\text{mean}() - m((x * x).\text{mean}())$$

$$- (y.\text{mean}() * x.\text{mean}())$$

$$- m * x.\text{mean}() * x.\text{mean}() = 0.$$

$$\Rightarrow \frac{(x * y).\text{mean}() - (y.\text{mean}() * x.\text{mean}())}{(x * x).\text{mean}() \overset{-}{\cancel{*}} x.\text{mean}() * x.\text{mean}()} = m$$

$$\Rightarrow \boxed{y.\text{mean}() - m * x.\text{mean}() = c}$$

These eqns give us the coefficients to find our best fitting line for our data.