KNN IMPUTER →

PROBLEM OF DEGREE OF FREEDOM :

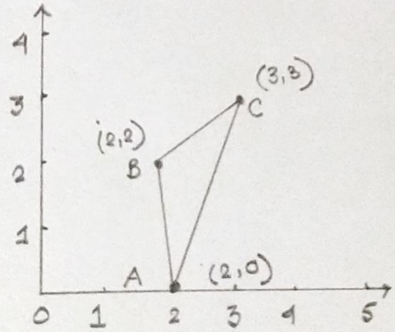
- Generally, if the proportion of missing observations in a dataset is small relative to number of observations, we can simply remove those observations However this is not often most cases.

- Deleting the rows containing missing values may lead to parting away with useful informations or patterns.
- In statistical terms, this leads to reduce degree of freedom as the number of independent pieces of information goes down.

KNN IMPUTATION –

- KNN identifies the neighbouring points through a measure of distance and the missing values can be estimated using value of neighbouring observations.
- Neighbouring points of a dataset are identified by certain distance metric generally euclidean distance.

DISTANCE CALCULATION IN THE PRESENCE OF MISSING VALUES –



Consider 3 observation A(2,0), B(2,2), C(3,3)

$d_{AB} = \sqrt{(2-2)^2+(0-2)^2} = \sqrt{0+4} = \sqrt{4}$

$d_{BC} = \sqrt{(3-2)^2+(3-2)^2} = \sqrt{1+1} = \sqrt{2}$

$d_{AC} = \sqrt{(2-3)^2+(0-3)^2} = \sqrt{1+9} = \sqrt{10}$

The point with the shortest distance based on Euclidean distances are considered to the nearest neighbour.

For example, 1 nearest neighbour to Point A is Point B. For point B, 1NN is C.

In the presence of missing coordinates, Euclidean distance is calculated by ignoring the missing values and scaling up the weight of non-missing coordinat

$$\boxed{d_{xy} = \sqrt{\text{weight} * \text{square distance from present coordinates}}}$$

where, $\text{weight} = \dfrac{\text{Total number of coordinates}}{\text{Number of present coordinates}}$

For example, two datapoints A(3, NA, 5) and B(1,0,0)

KNN Impution of NA = $\sqrt{\frac{3}{2}\{(3-1)^2+(5-0)^2\}} = 6.595$

In python,

```
import numpy as np.
from sklearn.metrics.pairwise import nan_euclidean_distances.
  X = [[3,np.nan, 5]]
  Y = [[1,0,0]]
 nan_euclidean_distances (X,Y)  # 6.595.
```

Suppose we have matrix/dataset

```
 from sklearn.impute import KNNImputer.
 import numpy as np.
  X = [[3,np.NaN, 5],[1,0,0],[3,3,3]]
 imputer = KNNImputer (n_neighbors =1).
 impute_with_1 = imputer.fit_transform (X)
 imputer = KNNImputer (n_neighbors =2).
 impute_with_2 = imputer.fit_transform (X).
```