KNN Imputer

17 July 2023 15:28

Multivariate Imputation:-

This includes **imputing the missing value** in any feature **based on the values in other observations.**

As we know there are two techniques available for Multivariate Imputation which are **kNN Imputation and Iterative Imputation (MICE)**.

---

kNN Imputer :

(k – Nearest Neighbors) where **k** – Number of Neighbors.

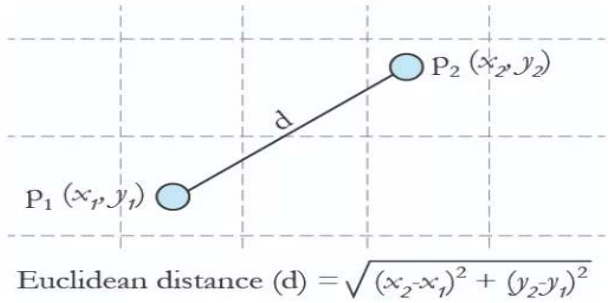
KNN Imputer **fills the missing value** in **a feature** in **a row** using values of **closest neighboring rows** in the **same feature**.

Simply, we can say **kNN Imputer** works on the **principle** that **all your characteristics and behavior** are **most likely to be similar** to that of **your closest neighbor** or **Average of your closest neighbors**.

In context of imputing missing values, We can say that kNN Imputer fills any missing value in a row using same feature values in similar or neighboring row/s.

And this **similarity of row/s** is calculated on the basis of **Euclidean Distance** by considering all the datapoints in an observation as coordinates of a point.

The **Euclidean Distance** is the **straight-line distance** between two points.



Two Major Steps in kNN Imputation:-

1. Finding out the k- Nearest Neighbours based on Euclidean Distances.
2. Calculating the Value to impute the missing value based on those k- Nearest Neighbours.

As there can be missing values in any feature of the row, so for two points it's very possible that values of all coordinates are not available. In that case instead of Euclidean Distance, we use **Nan_Euclidean_Distance.** So

Nan_Euclidean_Distance is calculated when some there are missing coordinate values in the observation.

Feature 2	Feature 3	Feature 4
-----	67	21
45	68	12
51	71	18
-----	81	------
60	79	------

Formula and Example for Nan_Euclidean_Distance:

dist(x,y) = sqrt(weight * sq. distance from present coordinates) where, weight = Total # of coordinates / # of present coordinates

For example, the distance between `[3, na, na, 6]` and `[1, na, 4, 5]` is:

$$\sqrt{\frac{4}{2}((3-1)^2+(6-5)^2)}$$

If all the coordinates are missing or if there are no common present coordinates then NaN is returned for that pair.

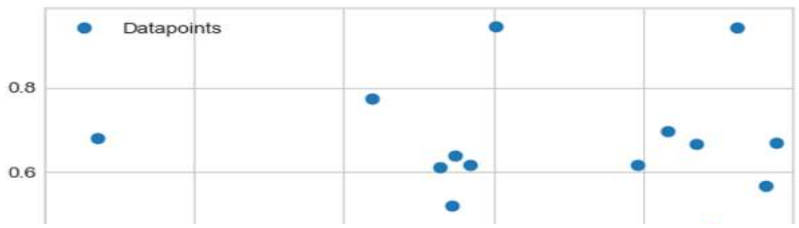
For Example in the image below, We want to fill missing value in Feature1. This we'll do on the basis of value in Nearest Neighbour/s.
Let's say all the values in 2nd row are Coordinates of a point.
I.e. P2(45, 68, 12) >> P2(x1, y1, z1)

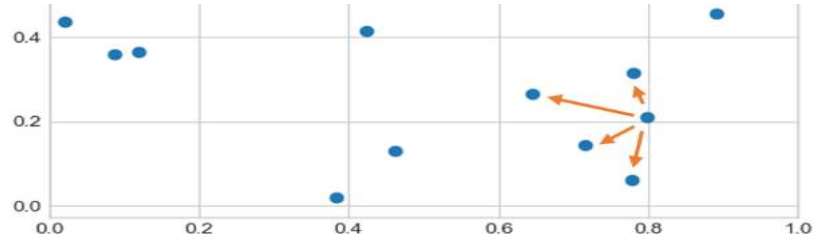
Now we'll **calculate Nan_Euclidean_Distance** of **P2** from **row1 [I.e. P1(nan, 67, 21)]** and **row4 [P4(nan, 81, nan)].**

S NO	Feature 1	Feature 2	Feature 3	Feature 4
1	33	-----	67	21
2	-----	45	68	12
3	23	51	71	18
4	40	-----	81	------
5	35	60	79	------

dist(x,y) = sqrt(weight * sq. distance from present coordinates) where, weight = Total # of coordinates / # of present coordinates

d (P1,P2) = **Sqrt [** 3/2 * **(** (67-68)^2 + (21-12)^2 **)]**
d (P1,P4) = **Sqrt [** 3/1 * **(** (81-68)^2 **)]**





Now if **n_neighbors = 1**, then we will impute the missing value with **Same Value in Nearest Neighbour row**.

If **n_neighbors > 1**, then we will impute the missing value with **Average of Values of Nearest Neighbours**.

Note:

1. The **nearest neighbor** has **minimum Nan_Euclidean_Distance** from the row to be imputed.
2. We can decide the number of neighboring rows to be considered for imputing the value by setting **n_neighbors** attribute in **KNNImputer.**
   Example: **knn = KNNImputer(n_neighbors=3)**.
3. If **n_neighbors** is greater than 1, then missing value will be filled with **Average of Nearest Neighbors Values in the same feature**.

When to Use:

1. When data is **MCAR** or **MAR**.
2. KNN imputer is useful when the missing data points are expected to be **similar to nearby observed data points**, as it relies on the **similarity of data points** in the feature space to impute the missing values like dataset telling about BMI based on Gender, Age , Height and Weight.

Advantages:

1. This technique is **more accurate** in terms of imputing missing values and giving better model accuracy.

Disadvantages:

1. This involves **lot of calculations**.
2. This is **memory heavy** as we have to store the training data on server for finding nearest neighbors while model is in production.

We should try to find out the optimum value for **n_neighbors** and **wieghts** parameters of kNN Imputer for better

performance of the model.

weights : *{'uniform', 'distance'} or callable, default='uniform'*

Weight function used in prediction. Possible values:

- 'uniform' : uniform weights. All points in each neighborhood are weighted equally.
- 'distance' : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.

Let's say, we have found out the Nan Euclidean Distances of points P2 (d = 2) and P3 (d = 2.5) from P1.

nan	**15**	**23**
50	**17**	**19**
55	**13**	**28**

1. Now in case of **weights = uniform:**
   Value (to fill missing value in P1) = **(50 + 55) / 2 = 52.5**

2. In case of **weights = distance:**
   Value (to fill missing value in P1) **= [(1/2)*(50) + (1/2.5)*(55)]/ 2**
   = 25 + 22 = 47

By using **weights = distance**, we can give **more importance** to **closer neighbors** for imputing missing values.