kNN-based imputation using Scikit-learn and Euclidean distance method.

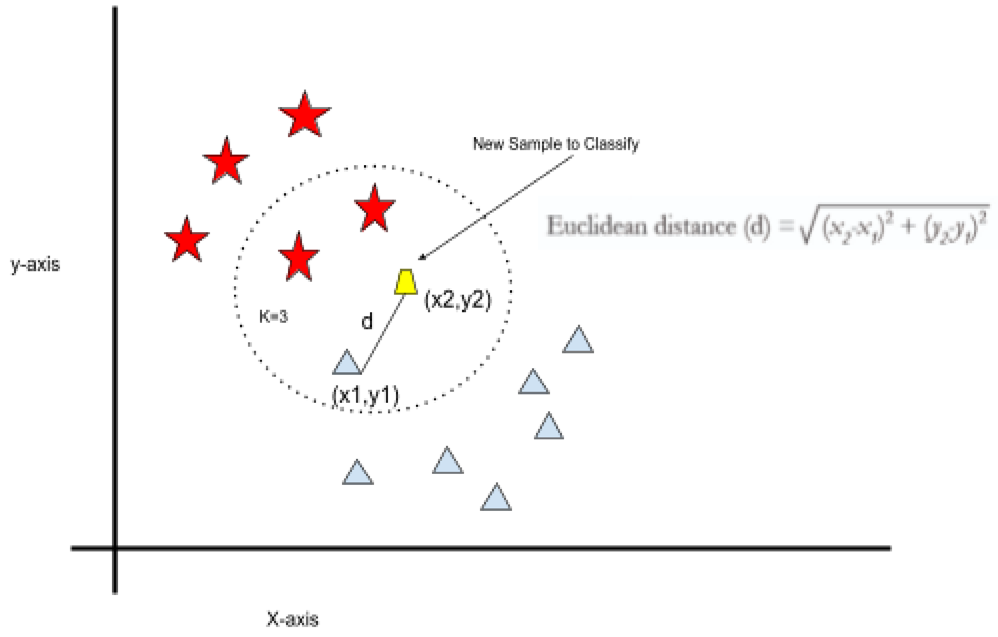
Euclidean distance:

Euclidean distance is the most popularly used distance measure. Euclidean distance also called as simply distance. The usage of Euclidean distance measure is highly recommended when data is dense or continuous kNN(K-Nearest Neighbor) imputation method used distance as Euclidean distance. Euclidean distance is the best proximity measure. The Euclidean distance between two points is the length of the path connecting them.The Pythagorean theorem gives this distance between two points.As we discussed the principle behind KNN imputer method is to find K predefined number closest in the distance to new point & predict the label from these. The distance measure is commonly considered to be Euclidean distance. kNN based imputation implementaion using Euclidean distance is as shown below.

kNN-Based Imputation

In the kNN imputation method, the missing values of an attribute are imputed using the attributes that are most similar to the attribute whose values are missing.

Imputation for completing missing values using k-Nearest Neighbors. Each sample's missing values are imputed using the mean value from n_neighbors nearest neighbors found in the training dataset. Two samples are close if the features that neither is missing are close.



The assumption behind using kNN for missing values is that a point value can be approximated by the values of the points that are closest to it, based on other variables.

The similarity of two attributes is determined using a distance function. Here are a few advantages of using kNN:

- The k-nearest neighbor can predict both qualitative & quantitative attributes
- Creation of predictive machine learning model for each attribute with missing data is not required
- Correlation structure of the data is taken into consideration

Parameters	**missing_values*int, float, str, np.nan or None, default=np.nan*** The placeholder for the missing values. All occurrences of missing_values will be imputed. For pandas' dataframes with nullable integer dtypes with missing values, missing_values should be set to np.nan, since pd.NA will be converted to np.nan.
	n_neighbors*int, default=5* Number of neighboring samples to use for imputation.
	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.

	● callable : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.
	metric*{'nan_euclidean'} or callable, default='nan_euclidean'* Distance metric for searching neighbors. Possible values: ● 'nan_euclidean' ● callable : a user-defined function which conforms to the definition of _pairwise_callable(X, Y, metric, **kwds). The function accepts two arrays, X and Y, and a missing_values keyword in kwds and returns a scalar distance value.
	copy*bool, default=True* If True, a copy of X will be created. If False, imputation will be done in-place whenever possible.
	add_indicator*bool, default=False* If True, a **MissingIndicator** transform will stack onto the output of the imputer's transform. This allows a predictive estimator to account for missingness despite imputation. If a feature has no missing values at fit/train time, the feature won't appear on the missing indicator even if there are missing values at transform/test time.
Attributes	**indicator_*`MissingIndicator`*** Indicator used to add binary indicators for missing values. `None` if add_indicator is False.

Scikit-learn supports kNN-based imputation using the Euclidean distance method. Let’s see a quick demo:

Import numpy and KNNImputer library and take sample dataset X with NaN values and imputed using KNNImputer as follows,

```
import numpy as np

from sklearn.impute import KNNImputer

X = [[6, 7, np.nan], [3, 4, 3], [5, 7, np.nan],[np.nan, 6, 5],
[6, 6, 9],[6, 8, np.nan]]

imputer = KNNImputer(n_neighbors=2)

print(imputer.fit_transform(X))
```

#Output:

```
[[6.  7.  7. ]

 [3.  4.  3. ]

 [5.  7.  7. ]

 [5.5 6.  5. ]

 [6.  6.  9. ]

 [6.  8.  7. ]]
```

Methods

fit(X[, y])	Fit the imputer on X.
fit_transform(X[, y])	Fit to data, then transform it.
get_params([deep])	Get parameters for this estimator.
set_params(**params)	Set the parameters of this estimator.

transform(X)	Impute all missing values in X.