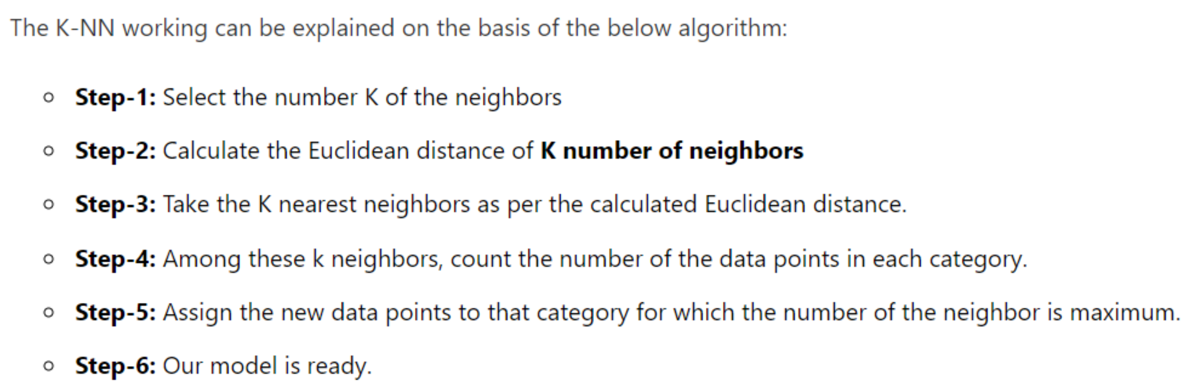
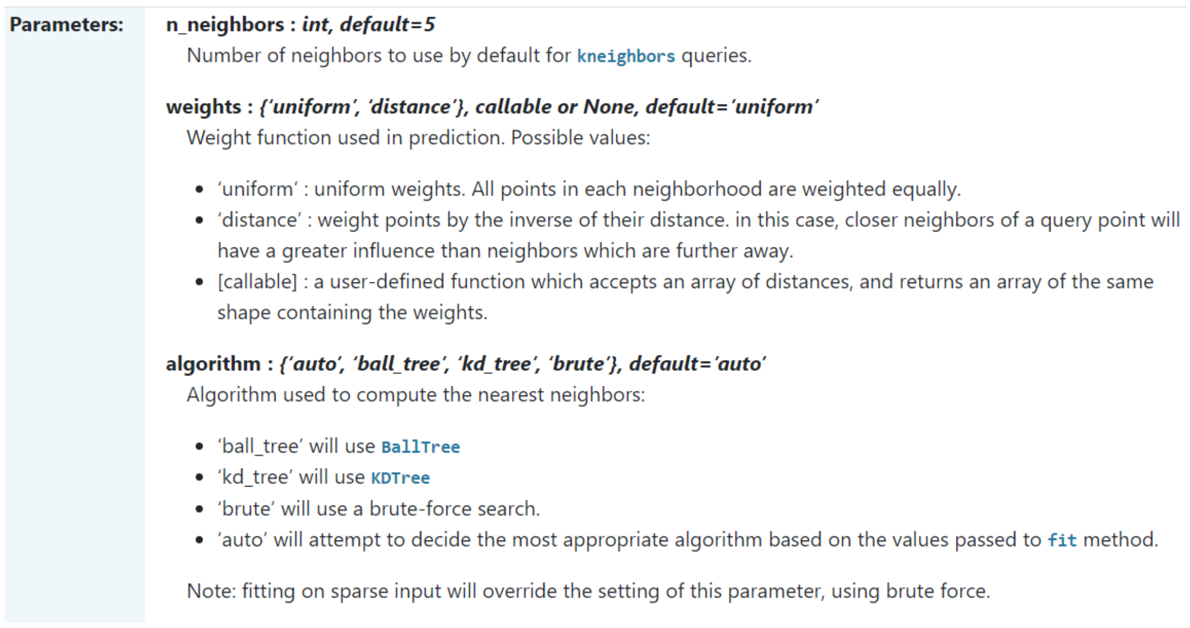
KNN Algorithm in Sleep Stage Prediction Model


The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

The parameters offered by sklearn.neighbors.KNeighborsClassifier are :-


Parameters:

n_neighbors : *int, default=5*
Number of neighbors to use by default for `kneighbors` queries.

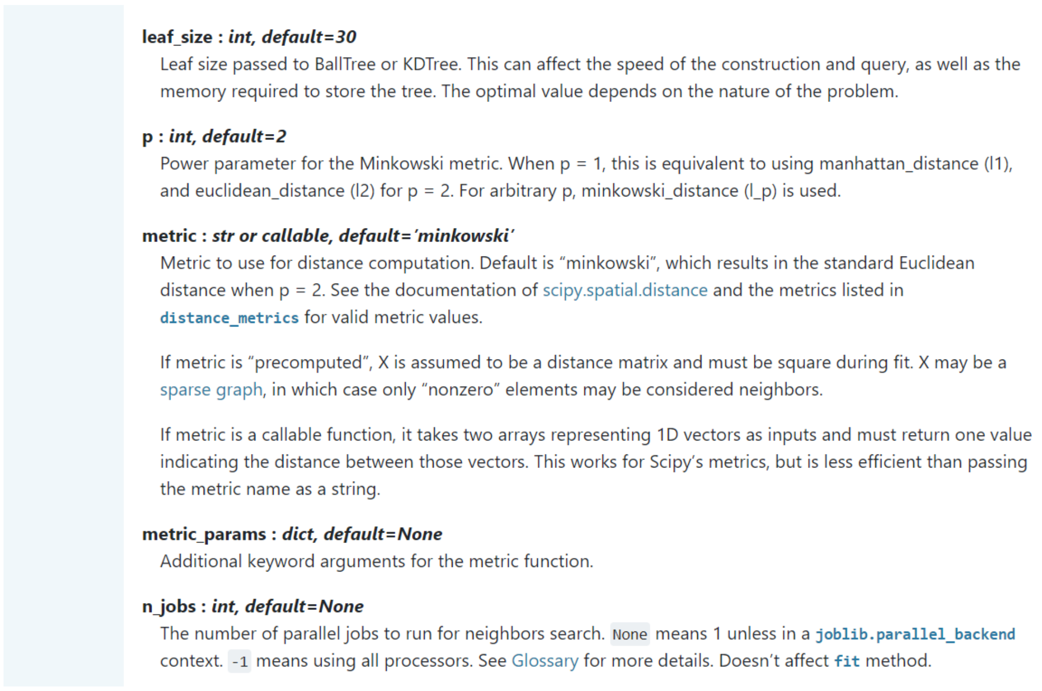
weights : *{'uniform', 'distance'}, callable or None, default='uniform'*
Weight function used in prediction. Possible values:

- '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.

algorithm : *{'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'*
Algorithm used to compute the nearest neighbors:

- 'ball_tree' will use `BallTree`
- 'kd_tree' will use `KDTree`
- 'brute' will use a brute-force search.
- 'auto' will attempt to decide the most appropriate algorithm based on the values passed to `fit` method.

Note: fitting on sparse input will override the setting of this parameter, using brute force.



leaf_size : *int, default=30*
Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.

p : *int, default=2*
Power parameter for the Minkowski metric. When p = 1, this is equivalent to using manhattan_distance (l1), and euclidean_distance (l2) for p = 2. For arbitrary p, minkowski_distance (l_p) is used.

metric : *str or callable, default='minkowski'*
Metric to use for distance computation. Default is "minkowski", which results in the standard Euclidean distance when p = 2. See the documentation of scipy.spatial.distance and the metrics listed in `distance_metrics` for valid metric values.

If metric is "precomputed", X is assumed to be a distance matrix and must be square during fit. X may be a sparse graph, in which case only "nonzero" elements may be considered neighbors.

If metric is a callable function, it takes two arrays representing 1D vectors as inputs and must return one value indicating the distance between those vectors. This works for Scipy's metrics, but is less efficient than passing the metric name as a string.

metric_params : *dict, default=None*
Additional keyword arguments for the metric function.

n_jobs : *int, default=None*
The number of parallel jobs to run for neighbors search. `None` means 1 unless in a `joblib.parallel_backend` context. `-1` means using all processors. See Glossary for more details. Doesn't affect `fit` method.

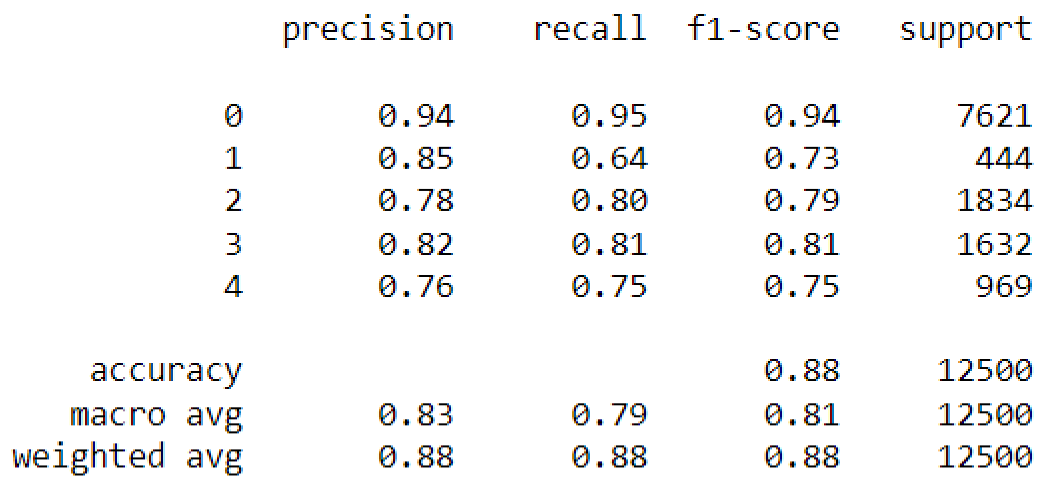
Architecture

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors=6,weights='distance',algorithm='ball_tree')
classifier.fit(X_train, Y_train)
```

The n_neighbors parameter determines the value of K. We have taken the value of n_neighbors = 6, as the total number of categories (Sleep Stages) is 5. The weights = 'distance' sets the criteria for selecting the K nearest neighbours as Euclidean Distance. The algorithm = 'ball_tree' sets the algorithm as Ball Tree algorithm. Setting all other parameters as default.

This architecture gives the best results in the KNN Classifier.

Classification Report



```
              precision    recall  f1-score   support

           0       0.94      0.95      0.94      7621
           1       0.85      0.64      0.73       444
           2       0.78      0.80      0.79      1834
           3       0.82      0.81      0.81      1632
           4       0.76      0.75      0.75       969

    accuracy                           0.88     12500
   macro avg       0.83      0.79      0.81     12500
weighted avg       0.88      0.88      0.88     12500
```