./smd/blatt10/knn.py
2021-07-06T14:08+02:00

```python
import numpy as np
from tqdm.notebook import tqdm

class KNN:
    '''KNN Classifier.

    Attributes
    ----------
    k : int
        Number of neighbors to consider.
    '''
    def __init__(self, k):
        '''Initialization.
        Parameters are stored as member variables/attributes.

        Parameters
        ----------
        k : int
            Number of neighbors to consider.
        '''
        self.k = k

    def fit(self, X, y):
        '''Fit routine.
        Training data is stored within object.

        Parameters
        ----------
        X : numpy.array, shape=(n_samples, n_attributes)
            Training data.
        y : numpy.array shape=(n_samples)
            Training labels.
        '''
        self.X_ = X
        self.y_ = y

    def predict(self, X):
        '''Prediction routine.
        Predict class association of each sample of X.

```

```
        Parameters
        ----------
        X : numpy.array, shape=(n_samples, n_attributes)
            Data to classify.

        Returns
        -------
        prediction : numpy.array, shape=(n_samples)
            Predictions, containing the predicted label of each sample.
        '''
        prediction = np.empty(shape=(X.shape[0]))
        for i,x in enumerate(tqdm(X)):
            prediction[i] = self.predict_single(x)

        return prediction

    def predict_single(self,x):
        '''Predict routine for a single sample
        Predict class association for a single sample x

        Parameters
        ----------
        x : numpy.array, shape=(n_attributes,)
            Data to classify.

        Returns
        -------
        prediction : int
            Prediction, containing the predicted label of the sample.
        '''
        x = x.reshape(1,-1)
        # Calculate all distances to every sample in the training
        ↪  dataset
        distances = np.linalg.norm(self.X_ - x, axis=1)

        # find the k nearest neighbors
        k_nearest = np.argsort(distances)[:self.k]

        # associate the best fitting class label
        y_unique, y_counts = np.unique(self.y_[k_nearest],
        ↪  return_counts=True)
        return y_unique[np.argmax(y_counts)]
```