Blatt10

July 13, 2020

1 Aufgabe 20

a) Worauf müssen Sie bei einem -NN-Algorithmus achten, wenn die Attribute sich stark in ihren GröSSenordnungen unterscheiden?

Unterscheiden sich die Attribute stark in ihrer GröSSenordnung, dann werden diese unterschiedlich stark gewichtet wodurch der Klassifizierer nicht mehr so genau arbeiten kann. Aus diesem Grund ist eine Normierung wichtig.

b) Warum bezeichnet man den -NN als sogenannten lazy learner"? Wie sind die Laufzeiten für Lern- und Anwendungs-Phase? Wie sind sie im Vergleich zu anderen Algorithmen wie bspw. einem Random Forest?

Der k-NN wird als "lazy learner" bezeichnet, weil der Algorithmus die Trainingsdaten nicht verwendet, um zu generalisieren, sondern den Trainingsdatensatz auswendig lernt. Die Anwendungsphase ist relativ lang und die Lernphase sehr kurz, weil Datensatz einfach auswendig gelernt wird.

c) Implementieren Sie einen -NN Algorithmus zur Klassifikation von Ereignissen. Halten sie sich hierbei an die in der beiliegenden Datei class_structure.py vorgegebenen Klassenstruktur. Die Methode predict soll dabei ein numpy array ausgeben, dass das vorhergesagte Label für jedes Sample enthält. Vorgehen: Für jedes zu klassifizierende Ereignis:

1) Berechnung der Abstände zu allen Punkten des Trainingssamples.
2) Bestimmung der Trainingsevents mit dem kleinsten Abstand (Hinweis: Ermitteln Sie nur die Indizes der Ereignisse, statt das Array an sich zu sortieren). Tipp: Die Python-Funktion numpy.argsort() ist hilfreich.
3) Bestimmung des Labels, das in diesen Ereignissen am häufigsten vorkommt.

```
In [1]: import numpy as np
        from scipy.spatial.distance import cdist

        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.train_data = X
        self.train_labels = 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.
        '''

        self.test_data = X

        predicted_labels = []

        for i in range(len(self.test_data)):
            background = 0
```

```
                signal = 0

                xx = self.test_data[i]

                distance = cdist(xx, self.train_data)[0]
                distance_list = np.stack((distance, self.train_labels), axis=1)
                #Eine Liste mit den Abständen der Trainingsdaten zu Testdaten und der zuge

                sorted_distance_list = distance_list[np.argsort(distance_list[:,0])] #List
                neighbours = sorted_distance_list[:self.k] #Bestimmung der k Trainingseven
                #Bestimmung des Labels, was am häufigsten vorkommt:
                for j in range(self.k):
                    if neighbours[j,1] == 0:
                        background += 1
                    else:
                        signal +=1
                if background >= signal:
                    predicted_labels.append(0)
                elif background < signal:
                    predicted_labels.append(1)

            return np.array(predicted_labels)
        #  print(np.array(predicted_labels))
```

In [ ]:

In [ ]: