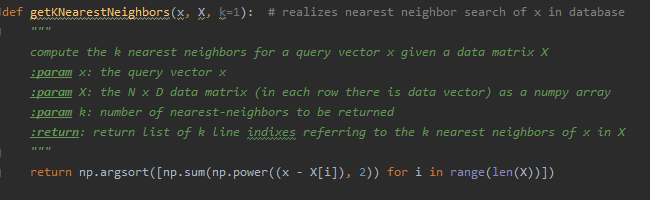
Abgabe Intelligente Adaptive Systeme Laborversuch 1

Malte Hoffmann

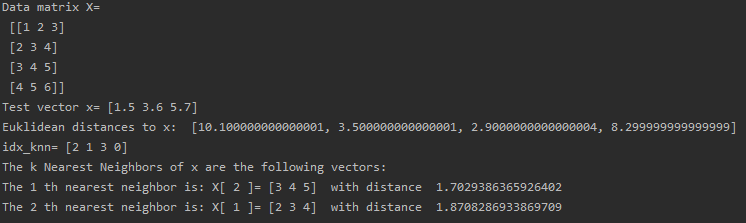
Tobias Schlauch

Steve Arnold Pouton Chedjou

V1A1

a)

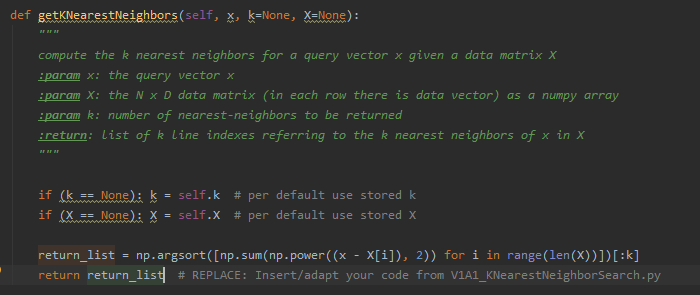
Für alle Datenvektoren d in X ziehen wir elementweise den Datenvektor x von d ab und quadrieren diese Differenz. Danach summieren wir alle Quadrate auf. Die daraus entstandene Liste sortieren wir mit argsort(), so dass wir den Index des kleinsten Elements an erster Stelle haben. Diese Liste geben wir zurück. Das kleinste Element hat hierbei den geringsten Abstand.

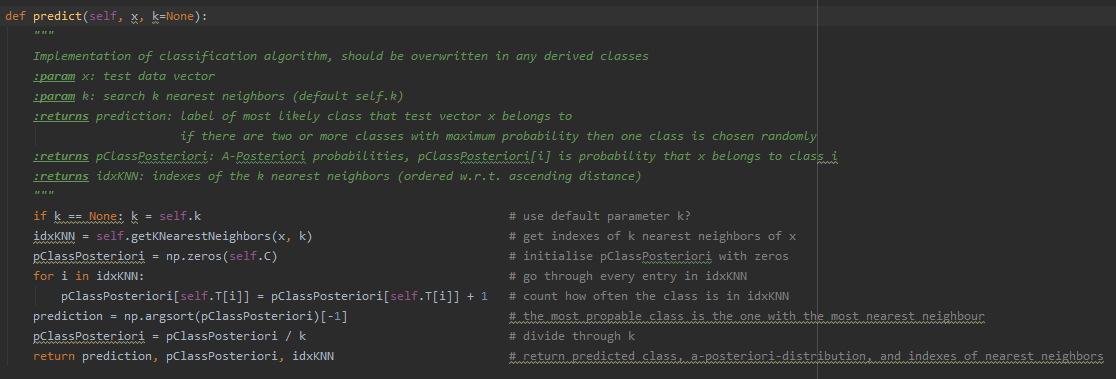
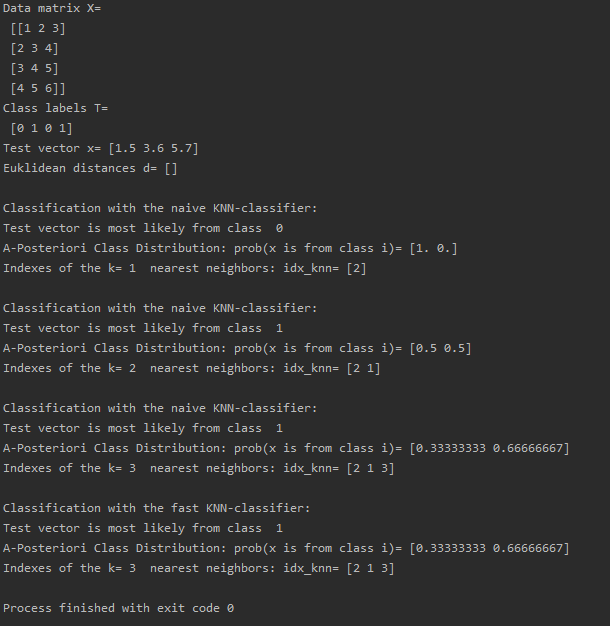
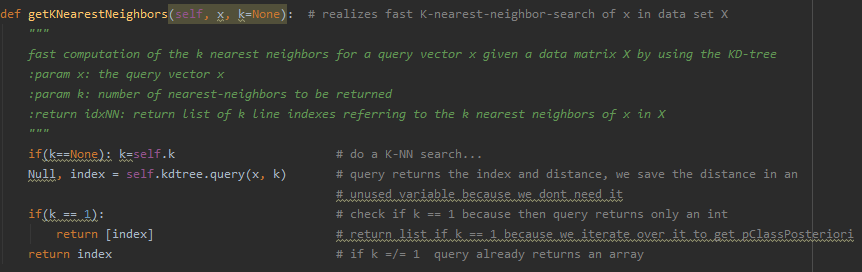
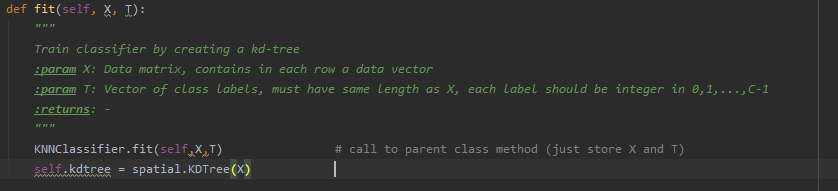
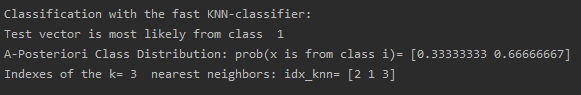
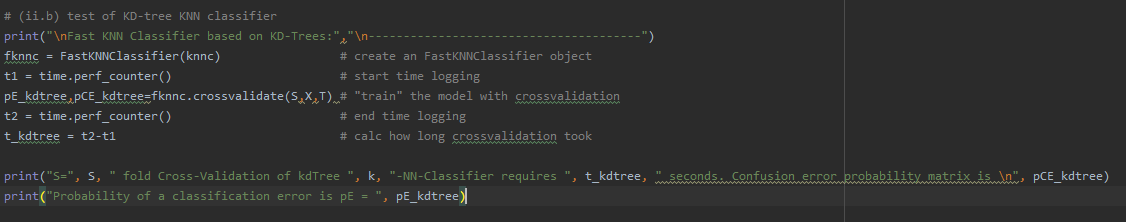
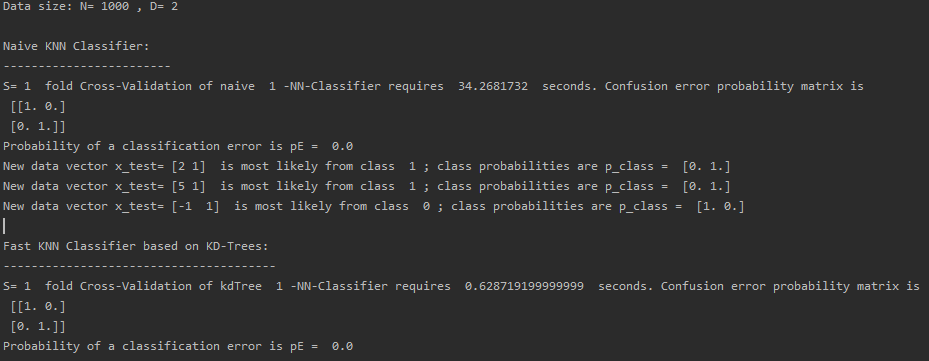
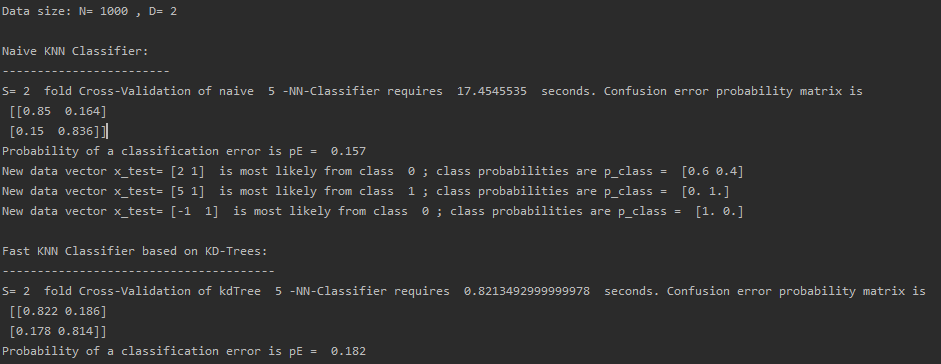
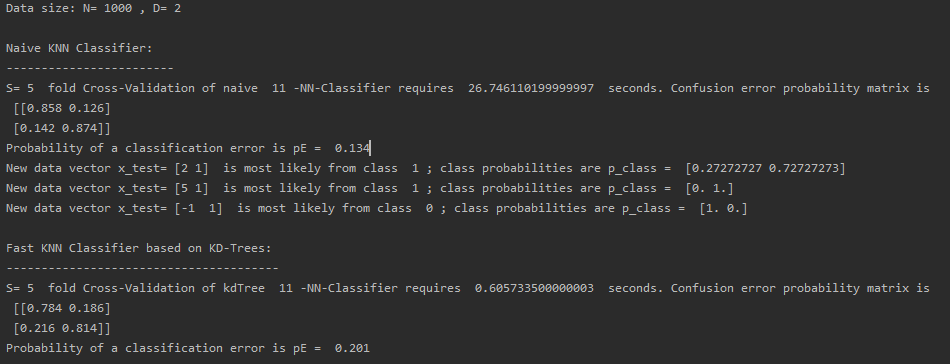
b)

Ausgabe unseres Codes von V1A1 a)

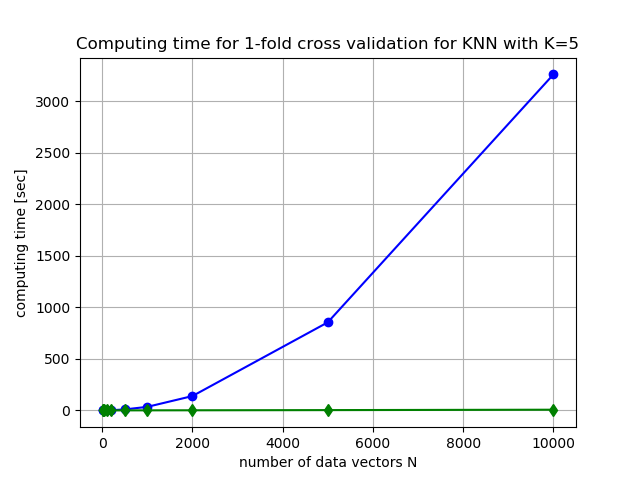
c)

sortieren ist n \* log(n)  
N := len(X), X = Datenmatrix -> Anzahl Datenvektoren => man braucht etwa O(1) Schritte mehr  
D := len(x), x = Datenvektor -> Anzahl Vektorelemente => man braucht etwa O(N) Schritte mehr  
k = Anzahl nearest neighbour => hat kaum Auswirkung auf die Laufzeit  
Schnellere Verfahren sind z.B. KD-Trees  
  
V1A2  
  
a)  
 Klassen:  
- Classifier, abstrakte Klasse, die das Grundgerüst und die zu implementierenden Funktionen angibt und eine eigene Funktion crossvalidate(self,S,X,T) implementiert  
- KNNClassifier, konkrete Klasse, die von Classifier erbt und alle Funktionen implementiert und zusätzlich eine eigene Funktion getKNearestNeighbors(self, x, k, X) implementiert  
- FastKNNClassifier, konkrete Klasse, die von KNNClassifier erbt und die fit(self,X,T) und getKNearestNeighbors(self, x, k=None) überschreibt. Diese müssen noch vervollständigt werden

Methoden:  
- \_\_init\_\_(self,C) -> Kontrsuktor der Klasse, wobei C die Anzahl der zu unterscheidenden Klassen angibt  
- fit(self,X,T) -> Funktion stellt sicher, dass die Datenmatrix ein zweidimensionales Array ist und der Datenvektor ein eindimensionales Array ist. Die Anzahl der Klassen wird in C gespeichert, allerdings nur wenn die Klassen durchgänig mit ganzen Integerzahlen durchnummeriert wurden  
- predict(self, x) -> muss von abgeleitenten Klassen implementiert werden. Soll die Wahrscheinlichkeiten, dass der neue Datenvektor x zu einer der Klassen aus T gehört ausgeben  
- crossvalidate(self,S,X,T) -> Teilt die Datenmenge in S Teile und trainiert das Model mit S-1 Teilen der Datenmenge, der S-Teil wird nach dem Trainieren zum Validieren benutzt. Am Ende wird die Wahrscheinlichkeit der Fehlklassifikation ausgegeben und eine Matrix in der angegeben ist mit welcher Wahrscheinlichkeit ein Objekt der Klasse j als ein Objekt der Klasse i klassifiziert wird.  
  
b) Die Klasse "lernt" nicht wirklich. Es wäre ein bereits trainiertes Netz, dass einfach nur in seiner Matrix nachschaut welcher bisherige Datenvektor am nächsten des neuen Datenvektor ist.  


Code-Erklärung siehe V1A1 a). Zusätzlich schneiden wir die Liste nach dem k-Eintrag ab.  
  
Codeerklärung siehe Kommentare im Code  
  
c)  
  
  
d) Man sollte für 2 Klassen immer ein ungerades k angeben, damit ausgeschlossen wird, dass gleiche viele neighbour aus den Klassen zurückgeliefert wird (z.B. 1 Neighbour aus Klasse 0 und 1 Neighbour aus Klasse 1).  
  
Verwendung der KDTree-Funktion aus dem Modul spatial  
Codeerklärung siehe Kommentare im Code  
Ausgabe unseres Programmes  
  
V1A3  
  
a)  
- 0%, da alle neighbours (=1) von einer Klasse sind  
- Nein, da es „Ausreiser“ geben kann. Also Datenvektoren die eigentliche im Bereich einer anderen Klasse liegen, dadurch kann es vorkommen, dass der Vektor mit dem kleinsten Abstand eine andere Klasse hat als alle Datenvektoren um ihn herum.  
- Man kann mit neuen Daten testen und über die Fehlerwahrscheinlichkeit den Mittelwert bilden. Es kann die Matrix mit der Fehlprognose-Wsk pro Klasse gebildet werden.  
- Kreuzvalidierung ist ein Verfahren zum Trainieren von neuronalen Netzten, dabei wird eine Datenmenge N in k-Teile aufgeteilt. Ein k-tel der Daten wird zur Validierung genutzt, der Rest zum Trainieren. Nach dem Training wird ein neues Model trainiert und ein anderer Teil der Daten wird zum Validieren benutzt. Hierdurch können alle Daten verwendet werden. Jedoch benötigt man viel Rechenpower/Zeit, da k Modelle trainiert werden müssen.  
  
b)  
-S = gibt an in wie viele Teile der Datensatz unterteilt werden soll  
-perm = Liste mit zufälliger Reihenfolge der Zahlen 0 bis N-1, jede Zahl kommt nur einmal vor  
-Xp = Liste mit Werten aus X, wobei die Sortierung von perm vorgegeben ist  
-Tp = Liste mit den Labels für die Vektoren in Xp mit gleicher Sortierung  
-idxS = Liste mit Indizes für die S-Teile der Größe N/S  
-for idxTest in idxS = Sucht den Teil aus, der als Validierungsdaten genutzt wird, sodass idxTest die Indizes der Validierungsdaten enthält. Es wird mit for durch die gesamte Datenmenge iteriert  
-X\_learn/X\_test = Indizes der Daten, die als Lerndaten verwendet werden (alle außer idxTest)  
-T\_learn /T\_test = Indizes der Labels für X\_learn, haben gleiche Sortierung  
-"S = 1" = idxLearn=idxTest -> es wird der gesamte Datensatz als Lerndaten verwendet und danach als Testdaten  
-for i in range(len(X\_test)) = über alle Datenvektoren in den Lerndaten iterieren  
-pClassError = addiert alle Fehler auf, wie oft ich insgesamt falsch lag  
-pConfErrors = Eine Matrix, die angibt mit welcher Wsk eine echte Klasse einer Klasse zugewiesen wird, z.Bsp. wenn ich in 3 Fällen Klasse 1 als 0 und in 1 Fall Klasse 0 als 1 vorhersage [9 3] [1 7]  
c)  
- Um alle Klassen, die wir in V1A2\_Classifier definiert und implementiert haben in unserem neuen Skript verwenden zu können  
- N1 und N2 sind die Datenvektoren mit denen gearbeitet wird. N1 ist von Klasse 0 und N2 von Klasse 1. N gibt die Anzahl der Spalten der Datenmatrix an, also wie viele Datenvektoren in der Matrix sind. D gibt die Dimensionalität der Datenvektoren an, bzw. wie viele Zeilen die Datenmatrix X hat.  
- die Datenvektoren sind Gaußverteilt mit den Mittelwerten mu1 = [1,1] und mu2 = [3,1]. Die Kovarinazmatrizen sind sigma1 = [[1,0.5], [0.5,1]] und sigma2 = [[1,0.5], [0.5,1]]   
- pE\_naive ist die Wahrscheinlichkeit einer Fehl-Klassifikation  
- pCE\_naive ist die Matrix, die die Wahrscheinlichkeit eine Objekt der Klasse a einer Klasse b zuzuordnen, hierbei kann b auch a sein  
- t\_naive ist die geschätzte Dauer zum Lernen des gesamten Datensatzes  
  
  
d)  
k = 1, S = 1  
  
  
k = 5, S = 2  
  
  
k = 11, S = 5  


|  |  |  |
| --- | --- | --- |
| k | Knnc | Fknnc |
| 1 | (2 1) = 1  (5 1) = 1  (-1 1) = 0 | (2 1) = 1  (5 1) = 1  (-1 1) = 0 |
| 5 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 |
| 11 | (2 1) = 1  (5 1) = 1  (-1 1) = 0 | (2 1) = 1  (5 1) = 1  (-1 1) = 0 |
| 111 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 |
| 511 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 | (2 1) = 0  (5 1) = 1  (-1 1) = 0 |

Die predicteten Klassen sind gleich, da der gleiche Algorithmus zur Berechnung angewandt wird.   
Allerdings ist die Fehlerwahrscheinlichkeit unterschiedlich, da bei der Kreuzvalidierung unterschiedliche Permutation erzeugt werden und damit die Modelle unterschiedlich „lernen“.  
  
e)  


|  |  |  |
| --- | --- | --- |
| N | T(N)knnc [sec] | T(N)fknnc [sec] |
| 10 | 0.006057800000000224 | 0.003429399999999916 |
| 20 | 0.017594200000000004 | 0.010788999999999938 |
| 50 | 0.10557680000000014 | 0.024012499999999992 |
| 100 | 0.38096629999999987 | 0.050906600000000246 |
| 200 | 1.5312214000000002 | 0.11193839999999966 |
| 500 | 9.5174417 | 0.2974475999999999 |
| 1000 | 34.2212492 | 0.6367327999999972 |
| 2000 | 138.96248359999998 | 1.1612240000000043 |
| 5000 | 854.7705021 | 2.9269702999999936 |
| 10000 | 3261.0793586000004 | 6.712470999999823 |