Intelligente adaptive Systeme

Nearest-Neighbor-Methoden für Klassifikation

Team:

Lisa-Marie Mai *87751*

Andreas Lay *87952*

Marius Schenzle *87937*

22.11.2019

Inhaltsverzeichnis

1 Implementierung einer k-Nearest-Neighbors-Suche in Python

1.1 Implementierung der Python-Funktion *getKNearestNeighbor(X, x, k=1)*

```
6
7   def getKNearestNeighbors(x, X, k=1):
8       """
9       compute the k nearest neighbors for a query vector x given a data matrix X
10      :param x: the query vector x
11      :param X: the N x D data matrix (in each row there is data vector) as a numpy array
12      :param k: number of nearest-neighbors to be returned
13      :return: return list of k line indixes referring to the k nearest neighbors of x in X
14      """
15
16      d = np.array([np.linalg.norm(X[i]-x) for i in range(len(X))])
17
18      return np.argsort(d)[0:k]
19
```

Abbildung 1: Python-Funktion *getKNearestNeighbor()*

1.2 Test der implementierten Python-Funktion

```
Data matrix X=
 [[1 2 3]
 [2 3 4]
 [3 4 5]
 [4 5 6]]
Test vector x= [1.5 3.6 5.7]
Euklidean distances to x:  [3.178049716414141, 1.8708286933869709, 1.7029386365926402, 2.8809720581775866]
idx_knn= [2 1]
The k Nearest Neighbors of x are the following vectors:
The 1 th nearest neighbor is: X[ 2 ]= [3 4 5]  with distance  1.7029386365926402
The 2 th nearest neighbor is: X[ 1 ]= [2 3 4]  with distance  1.8708286933869709
```

Abbildung 2: Test der Funktion

1.3 Angabe der Rechenschritte in O-Notation

Es muss für jeden Vektor in X die euklidische Distanz bestimmt werden. D. h. n Durchläufe. Weiterhin muss die Funktion *np.argsort()* die zuvor ermittelten euklidischen Distanzen sortieren, um den Index der kleinsten zurückzuliefern. Daraus folgt: $O(n + n * log(n))$ unter Annahme, dass *np.argsort()* mit Merge-Sort oder Heap-Sort aufgerufen wird, und die Laufzeit somit $n * log(n)$ beträgt.
Ein schnelleres Verfahren ginge über den KD-Tree.

2 Python-Modul für k-Nearest-Neighbor-Klassifikatoren

2.1 Aufbau des Moduls *V1A2_Classifier.py*

2.1.1 Klassen im Modul

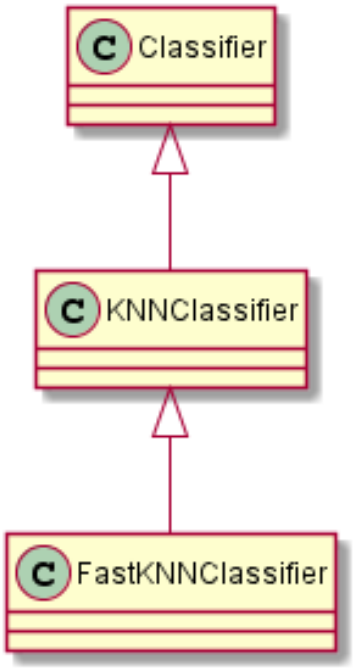


Abbildung 3: Übersicht Klassen in V1A2_Classifier.py

2.1.2 Betrachtung der Basis-Klasse *Classifier*

__init__(self, C): Ist der Konstruktor der Klasse
fit(self, X, T) prüft, ob die Matrizen *X(Trainingsdaten)* und *T(Klassenlabels)* die richtigen Dimensionen haben und speichert die Anzahl der Labels in *self.C*.
predict(self, x): Abstrakte Methode, die von den abgeleiteten Klassen implementiert werden muss, um die naheliegenste Klasse herauszufinden.
crossvalidate(self, S, X, T): Berechnet die Verwechslungsmatrix und die Chance für einen Klassifizierungsfehler.

2.2 Die Klasse *KNNClassifier*

2.2.1 Wie lernt ein k-NN-Klassifikator?

Der k-NN Klassifikator bestimmt die euklidische Distanz der k nächsten, bereits klassifizierten Nachbarn zu einem Merkmalsvektor. Der Merkmalsvektor wird der Klasse zugewiesen, welche unter der k Nachbarn am häufigsten vorkommt.

fit(self, X, T): Ruft die fit-Methode der Basisklasse auf. Wenn korrekt, weist diese Methode die Matrizen X und T den Klassenattributen zu.

2.2.2 Implementierung der Methode *getKNearestNeighbors(self, x, k=None, X=None)*

```
147     def getKNearestNeighbors(self, x, k=None, X=None):
148         """
149         compute the k nearest neighbors for a query vector x given a data matrix X
150         :param x: the query vector x
151         :param X: the N x D data matrix (in each row there is data vector) as a numpy
152         :param k: number of nearest-neighbors to be returned
153         :return: list of k line indexes referring to the k nearest neighbors of x in
154         """
155         if(k == None):
156             k = self.k                  # per default use stored k      You, 20 d
157         if(X == None):
158             X = self.X                  # per default use stored X
159 
160         d = np.array([np.linalg.norm(X[i]-x) for i in range(len(X))])
161 
162         return np.argsort(d)[0:k]
163 
```

Abbildung 4: Implementierung

2.2.3 Implementierung der Methode *predict(self, x, k=None)*

```
171         :returns pClassPosteriori: A-Posteriori probabilities, pClassPosteriori[i] is
172         :returns idxKNN: indexes of the k nearest neighbors (ordered w.r.t. ascending
173         """
174         if k == None:
175             k = self.k                      # use default parameter k?
176         # get indexes of k nearest neighbors of x
177         idxKNN = self.getKNearestNeighbors(x, k)
178         # get Classlabels by index
179         nearestClasslabels = self.T[idxKNN]
180 
181         # get sorted list of occuring classes and amount of occurrences
182         Class, count = np.unique(nearestClasslabels, return_counts=True)
183 
184         # get index(indces) of highest occurence(s)
185         idx_highest_occ = np.argwhere(count == np.amax(count))
186 
187         if len(idx_highest_occ) == 1:
188             idx_highest_occ = idx_highest_occ[0][0]
189         else:
190             selector = random.randint(0, len(idx_highest_occ)-1)
191             idx_highest_occ = idx_highest_occ[selector][0]
192 
193         # class with highest occurence
194         prediction = Class[idx_highest_occ]
195         # list with all Classes
196         All_Classes = range(self.C)
197         # chance that x is part of each Class
198         distrib = [round(count[list(Class).index(i)]/k, 4)
199                    if i in Class else 0 for i in range(self.C)]
200         # zip together in Dictionary
201         pClassPosteriori = dict(zip(All_Classes, distrib))
202         # return predicted class, a-posteriori-distribution, and indexes of nearest
203         return prediction, pClassPosteriori, idxKNN
204 
205 
```

Abbildung 5: Implementierung

2.3 Test der vollständig implementierten Klasse KNNClassifier

2.3.1 Ergebnisse der Tests

```
Classification with the naive KNN-classifier:
Test vector is most likely from class  0
A-Posteriori Class Distribution: prob(x is from class i)= {0: 1.0, 1: 0}
Indexes of the k= 1  nearest neighbors: idx_knn= [2]
The 1 th nearest neighbor is: X[ 2 ]= [3 4 5]

Classification with the naive KNN-classifier:
Test vector is most likely from class  1
A-Posteriori Class Distribution: prob(x is from class i)= {0: 0.5, 1: 0.5}
Indexes of the k= 2  nearest neighbors: idx_knn= [2 1]
The 1 th nearest neighbor is: X[ 2 ]= [3 4 5]
The 2 th nearest neighbor is: X[ 1 ]= [2 3 4]

Classification with the naive KNN-classifier:
Test vector is most likely from class  1
A-Posteriori Class Distribution: prob(x is from class i)= {0: 0.3333, 1: 0.6667}
Indexes of the k= 3  nearest neighbors: idx_knn= [2 1 3]
The 1 th nearest neighbor is: X[ 2 ]= [3 4 5]
The 2 th nearest neighbor is: X[ 1 ]= [2 3 4]
The 3 th nearest neighbor is: X[ 3 ]= [4 5 6]
```

Abbildung 6: Testergebnisse

2.3.2 Warum sollte man für C = 2 Klassen immer ungerades k wählen?

Da wenn man bei C = 2 z. B. k = 2 wählt, es sein kann, dass einer der Nachbarn Label 0 und der andere Label 1 hat und der Merkmalsvektor somit beiden Klassen zu 50% zugeordnet wird. Bei ungeradem k wird somit eine klare Klassentrennung gewährleistet.

2.4 Vervollständigung der von *KNNClassifier* abgeleiteten Klasse *FastKNNClassifier*

2.4.1 Die Methode *fit(self, X, T)*

```
224     def fit(self, X, T):
225         """
226         Train classifier by creating a kd-tree
227         :param X: Data matrix, contains in each row a data vector
228         :param T: Vector of class labels, must have same length as X, each label shou
229         :returns: -
230         """
231         # call to parent class method (just store X and T)
232         KNNClassifier.fit(self, X, T)
233 
234         self.kdtree = scipy.spatial.KDTree(X)
235 
```

Abbildung 7: Implementierung

2.4.2 Die Methode *getKNearestNeighbors(self, x, k=None)*

```
237     def getKNearestNeighbors(self, x, k=None):
238         """
239         fast computation of the k nearest neighbors for a query vector x given a data
240         :param x: the query vector x
241         :param k: number of nearest-neighbors to be returned
242         :return idxNN: return list of k line indexes referring to the k nearest neighl
243         """
244         if(k == None):
245             k = self.k                      # do a K-NN search...
246 
247         NULL, idxNN = self.kdtree.query(x, k)
248         return idxNN                                # return indexes of k nearest neig
249 
```

Abbildung 8: Implementierung

2.4.3 Test der Klasse *FastKNNClassifier*

```
284     # (iv) Repeat steps (ii) and (iii) for the FastKNNClassifier (based on KD-Trees)
285 
286     fknnc = FastKNNClassifier()     # construct FastkNN Classifier
287     fknnc.fit(X, T)                 # train with given data
288 
289     # (iii) Classify test vector x
290     k = 3
291     c, pc, idx_knn = fknnc.predict(x, k)
292     print("\nClassification with the FastKNN-classifier:")
293     print("Test vector is most likely from class ", c)
294     print("A-Posteriori Class Distribution: prob(x is from class i)=", pc)
295     print("Indexes of the k=", k, " nearest neighbors: idx_knn=", idx_knn)
296     for i in range(k):
297         print("The", i+1, "th nearest neighbor is: X[", idx_knn[i], "]=",
298                 X[idx_knn[i]])
299 
```

```
PROBLEME 1   AUSGABE   DEBUGGING-KONSOLE   TERMINAL

Classification with the FastKNN-classifier:
Test vector is most likely from class  1
A-Posteriori Class Distribution: prob(x is from class i)= {0: 0.3333, 1: 0.6667}
Indexes of the k= 3  nearest neighbors: idx_knn= [2 1 3]
The 1 th nearest neighbor is: X[ 2 ]= [3 4 5]
The 2 th nearest neighbor is: X[ 1 ]= [2 3 4]
The 3 th nearest neighbor is: X[ 3 ]= [4 5 6]
```

Abbildung 9: Testergebnisse

3 Kreuzvalidierung und Effizienz des k-NN-Klassifikators

3.1 Allgemeine Fragen zur Evaluation eines Klassifikators

3.1.1 Klassifikationsfehlerwahrscheinlichkeiten

Die Klassifikationsfehlerwahrscheinlichkeit ist 0, da es nur einen nächsten Nachbarn gibt und somit nur eine Klasse, die zugewiesen wird.

3.1.2 Diskussion des Ergebnis der Klassifikationsfehlerwahrscheinlichkeit

Ja, da diese auch immer nur einen Nachbar haben und somit nur einr Klasse zugeordnet werden können.

3.1.3 Möglichkeiten, um einen realistischen Schätzwert des Generalisierungsfehlers zu erhalten

Man kann für k einen größeren Wert wählen.

3.1.4 Begriff Kreuzvalidierung

Die Daten werden in S Teile aufgeteilt von denen immer ein Teil zur Validierung zurückgehalten wird. Dabei wird das Modell S mal trainiert. Am Ende erhält man dadurch den resultierenden Generalisierungsfehler.

3.2 Code-Review der Methode *Classifier.crossvalidate(self, S, X, T)*

Was bedeutet der Parameter S?

Die Daten werden in S Daten aufgeteilt. Es wird immer ein Teil zum validieren zurückgehalten.

Welche Rolle spielen die Variablen perm sowie Xp und Tp?

perm: eine zufällige Permutation der Zahlen 0 bis N-1 wird in einem np.array abgelegt. Liefert zufällige Indizes für Xp und Tp. *Xp und Tp*: Daten aus X werden in zufälliger Reihenfolge in Xp abgelegt. Daten aus T werden in zufälliger Reihenfolge n Tp abgelegt.

Welche Rolle spielt idxS?

Was bewirkt die äußere Schleife for idxTest in idxS? Durchläuft alle möglichen Datasets und holt sich die Indizes für Trainingsdaten und speichert diese in *X_learn* und *T_learn*. Die restlichen Daten sind Testdaten und werden in *X_test* und *T_test* gespeichert.

Was passiert für S=1?

Für S=1 wird das gesamte Dataset zum Lernen und Trainieren verwendet.

Was bewirkt die innere Schleife for i in range(len(X_test)): ...?

Durchläuft alle Datenvektoren in *X_test* und klassifiziert diese, holt sich aus *T_test* das zugehörige „richtige“ Label und schreibt in die Matrix nC die Häufigkeit, wie oft das jeweilige Label aufkommt. Später wird die Anzahl der Fehler ausgewertet.

Was bedeuten die Ergebnisse der Kreuzvalidierung pClassError und pConfErrors?

In pClassError wird aufsummiert, wie oft die Testdaten nicht korrekt klassifiziert wurden. Die Anzahl wird dann noch durch die Gesamtheit aller Datenvektoren geteilt, um die Wahrscheinlichkeit eines Klassifikationsfehlers zu erhalten. In der Matrix pConfErrors ist auf der Hauptdiagonalen die Anzahl der Fälle, in denen die Testdaten korrekt klassifiziert wurden. Alle anderen Elemente der Matrix enthalten die Anzahl der Fälle, in denen die Testdaten nicht korrekt klassifiziert wurden. Alle Elemente der Matrix werden dann noch durch die Wahrscheinlichkeit für das Auftreten der jeweiligen Klasse geteilt. So erhält man die Wahrheitsmatrix.

3.3 Betrachtung eines 2-Klassen-Problems für Gauß-verteilte Datenvektoren $x_n \in \mathbb{R}^D$ mit D-dim. Dichtefunktion

Wozu benötigt man den Befehl *from V1A2_ Classifier import* $*$?

Um die Funktionen die in *V1A2_ Classifier.py* definiert und implementiert wurden, in *V1A3_ CrossVal_ KNN.py* nutzen zu können.

Mit welchem Befehl werden die Gauß-verteilten Datenvektoren erzeugt? Wie viele Datenpunkte werden generiert? Was bedeuten die Variablen N, N1, N2?

Mit dem Befehl $numpy.random.multivariate_normal(mean, cov[, size, check_valid, tol])$ N1 und N2: Anzahl der Datenvektoren für die zwei Klassen. N: Anzahl der Datenvektoren von X1 und X2 insgesamt, also 1000.

Welche klassenspezifischen Verteilungen haben die Daten?

Klasse 1 Mittelwert: [1,1] Klasse 2 Mittelwert: [3,1] Klasse 1 Kovarianzmatrix: [[1,0.5] [0.5,1]] Klasse 2 Kovarianzmatrix: [[1,0.5] [0.5,1]]

Welche Bedeutung haben die Variablen pE_naive, pCE_naive und t_naive?

pE_naive: Wahrscheinlichkeit eines Klassifikationsfehlers
pCE_naive: Wahrheitsmatrix
t_naive: Berechnungszeit

```
# (ii.b) test of KD-tree KNN classifier
print("\nFast KNN Classifier based on KD-Trees:",
      "\n-------------------------------------")

fknnc = FastKNNClassifier(C, k)
t1 = clock()

pE_kdtree, pCE_kdtree = fknnc.crossvalidate(S, X, T)
t2 = clock()

t_kdtree = t2-t1

print("S=", S, " fold Cross-Validation of naive ", k, "-NN-Classifier requires ",
      t_kdtree, " seconds. Confusion error probability matrix is \n", pCE_kdtree)
print("Probability of a classification error is pE = ", pE_kdtree)
# train classifier with whole data set
fknnc.fit(X, T)
for x_test in X_test:            # Test some additional data vectors x_test from X_tes
    t_test, p_class, idxNN = fknnc.predict(x_test, k)
    print("New data vector x_test=", x_test, " is most likely from class ",
          t_test, "; class probabilities are p_class = ", p_class)
```

Abbildung 10: Codevervollständigung

3.4 Test der Kreuzvalidierung bzw. Klassifikationsleistung

3.4.1 Bestimmung der Klassifikationsfehler und Verwechselwahrscheinlichkeiten

3d) 1							
k	S	KNN			FastKNN		
		Classerror	Confusion Matrix		Classerror	Confusion Matrix	
1	1	0	1	0	0	1	0
			0	1		0	1
1	2	0.188	0.83	0.156	0.188	0.812	0.188
			0.17	0.844		0.188	0.812
1	5	0.188	0.826	0.202	0.169	0.836	0.174
			0.174	0.798		0.164	0.826
5	1	0.095	0.896	0.086	0.095	0.896	0.086
			0.104	0.914		0.104	0.914
5	2	0.141	0.866	0.148	0.143	0.848	0.134
			0.134	0.852		0.152	0.866
5	5	0.134	0.854	0.122	0.121	0.866	0.108
			0.146	0.878		0.134	0.892
11	1	0.114	0.87	0.098	0.114	0.87	0.098
			0.13	0.902		0.13	0.902
11	2	0.139	0.868	0.146	0.136	0.848	0.12
			0.132	0.854		0.152	0.88
11	5	0.134	0.858	0.126	0.13	0.858	0.118
			0.142	0.874		0.142	0.882

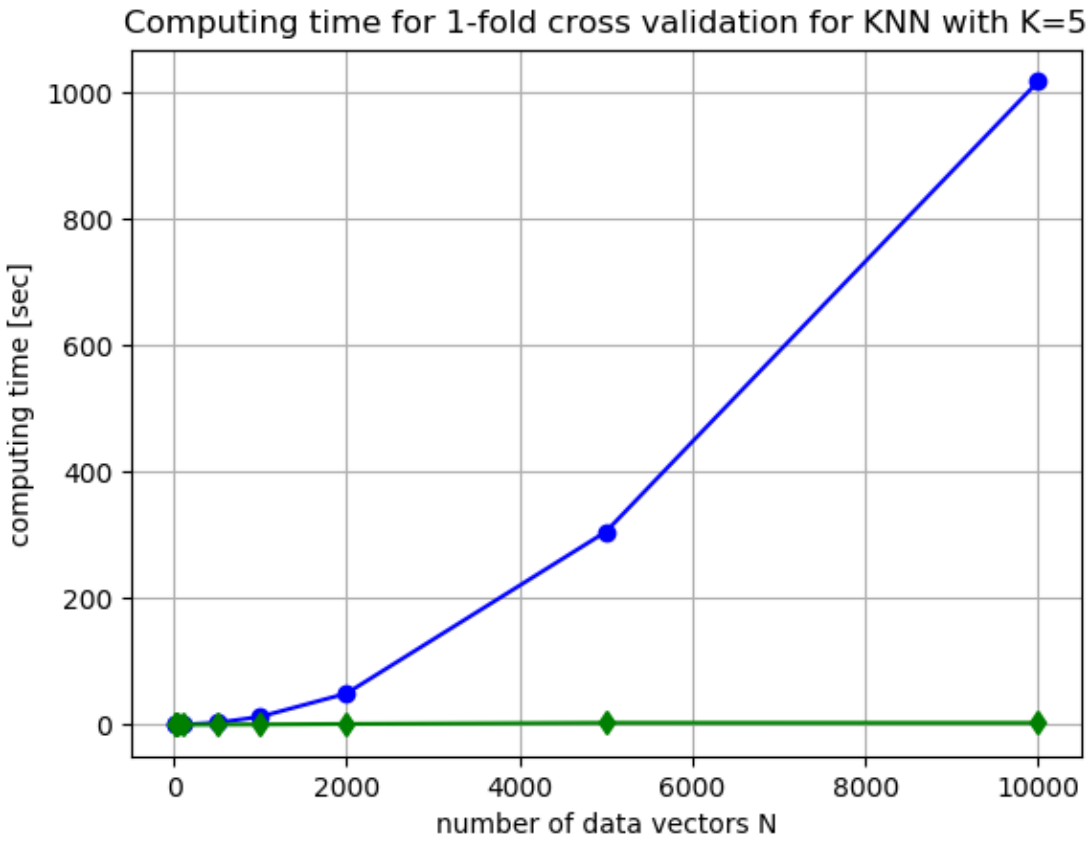
Abbildung 11: Wahrheitsmatrix

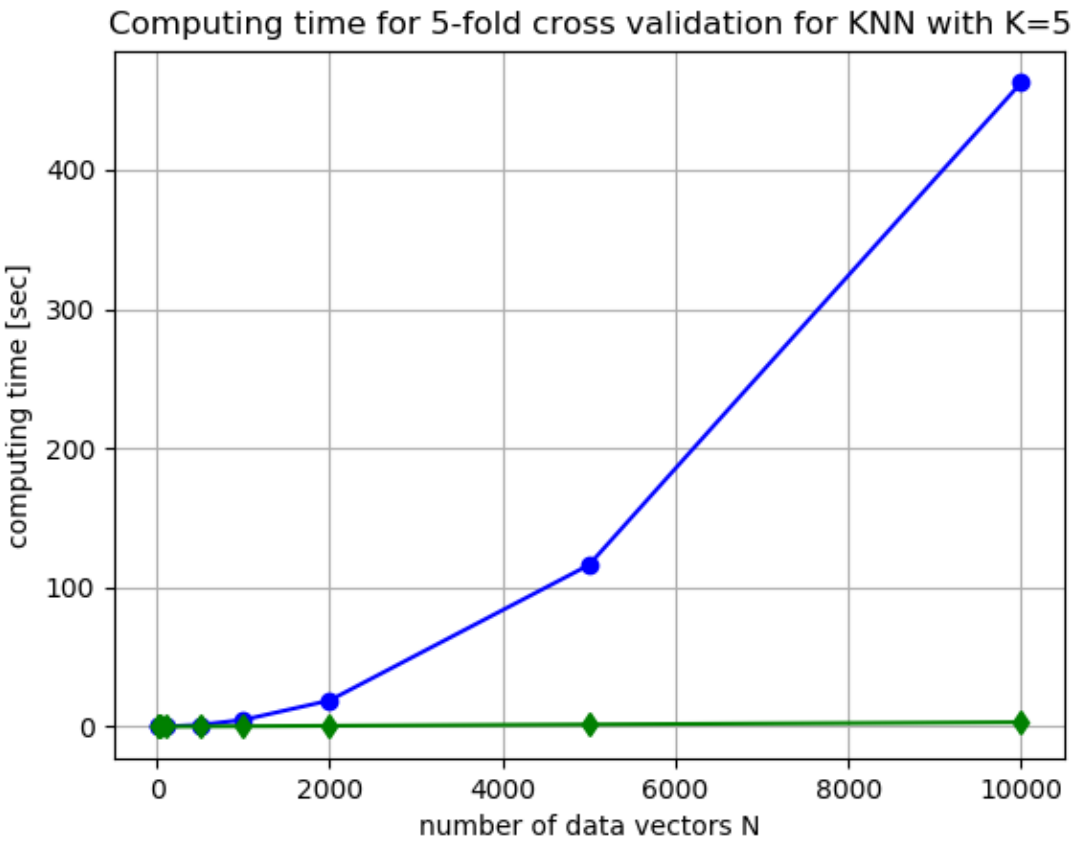
3.4.2 Bestimmung der Klassenverteilung für drei weitere Testpunkte

3 d) 2			
	KNN		
k	T1(2,1)	T2(5,1)	T3(-1,1)
1	{0:1, 1:0}	{0:0, 1:1}	{0:1, 1:0}
5	{0:0.8, 1:0.2}	{0:0, 1:1}	{0:1, 1:0}
11	{0:0.4545, 1:0.5455}	{0:0, 1:1}	{0:1, 1:0}
111	{0:0.5045, 1:0.4955}	{0:0, 1:1}	{0:0.991, 1:0.009}
511	{0:0.4912, 1:0.5088}	{0:0.1624, 1:0.8376}	{0:0.8356, 1:0.1644}

Abbildung 12: Klassenwahrscheinlichkeiten

3.5 Vergleich der Effizienz beider k-NN-Klassifikatoren




Computing time for 5-fold cross validation for KNN with K=5
computing time [sec]
0
100
200
300
400
0
2000
4000
6000
8000
10000
number of data vectors N