

Ausarbeitung Versuch 1 ILS Jan Holderied und Martin Goien

Aufgabe 1

b)

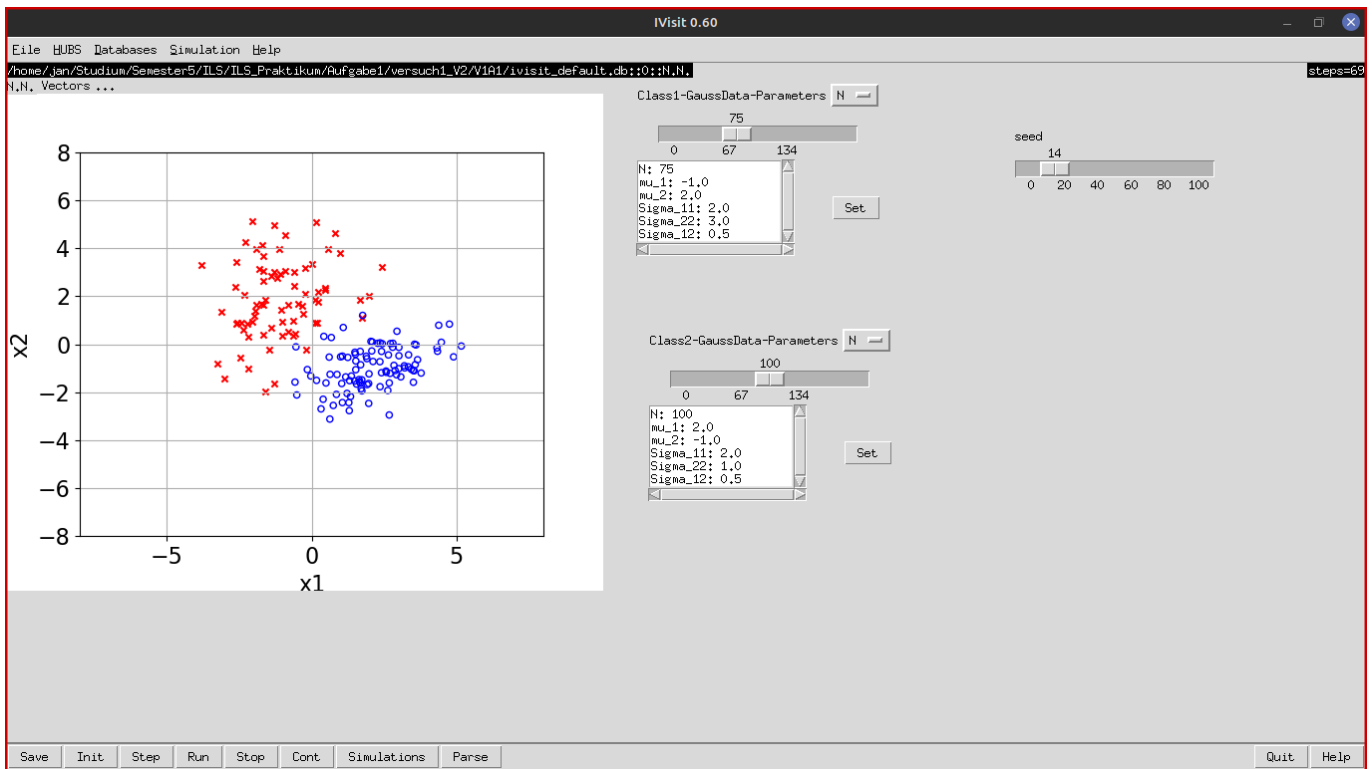
```

(base) jan@jan-P6640:~/Studium/Semester5$ /home/jan/anaconda3/bin/python /home/jan/Studium/Semester5/ILS/ILS_Pr
aktikum/Aufgabe1/versuch1_V2/V1A1/GaussDataGeneration.py
X= [[ 1.64685570e+00  9.21046021e-01]
 [ 1.44116497e+00  1.89291504e+00]
 [ 1.71502560e+00  3.84548876e+00]
 [ 2.04138475e+00  3.82041885e+00]
 [ 1.71442751e-01  4.18864319e+00]
 [ 2.85900388e+00  1.91811789e+00]
 [ 1.97861399e+00  1.44704790e+00]
 [ 1.54504118e-02  1.14938757e+00]
 [-5.26634217e-02  1.46467340e+00]
 [ 1.20720554e-01  -4.05798895e-01]
 [ 3.34010536e+00  -1.72174190e-03]
 [ 1.25600641e+00  6.13741513e-01]
 [ 6.60085356e-01  1.05885328e+00]]
T= [1. 1. 1. 1. 1. 2. 2. 2. 2. 2. 2.]
(base) jan@jan-P6640:~/Studium/Semester5$
  
```

Formeln um Mittelwert und Kovarianz zu schätzen. Siehe im Pythonskript die Funktionen ApproxMean und ApproxCovariance.

$$\mu \approx \tilde{\mu} := \frac{1}{n} \sum_{k=1}^n \mathbf{X}_k \quad \text{und} \quad \Sigma \approx \frac{1}{n-1} \sum_{k=1}^n (\mathbf{X}_k - \tilde{\mu}) \cdot (\mathbf{X}_k - \tilde{\mu})^T$$

c)



Aufgabe 2

d)

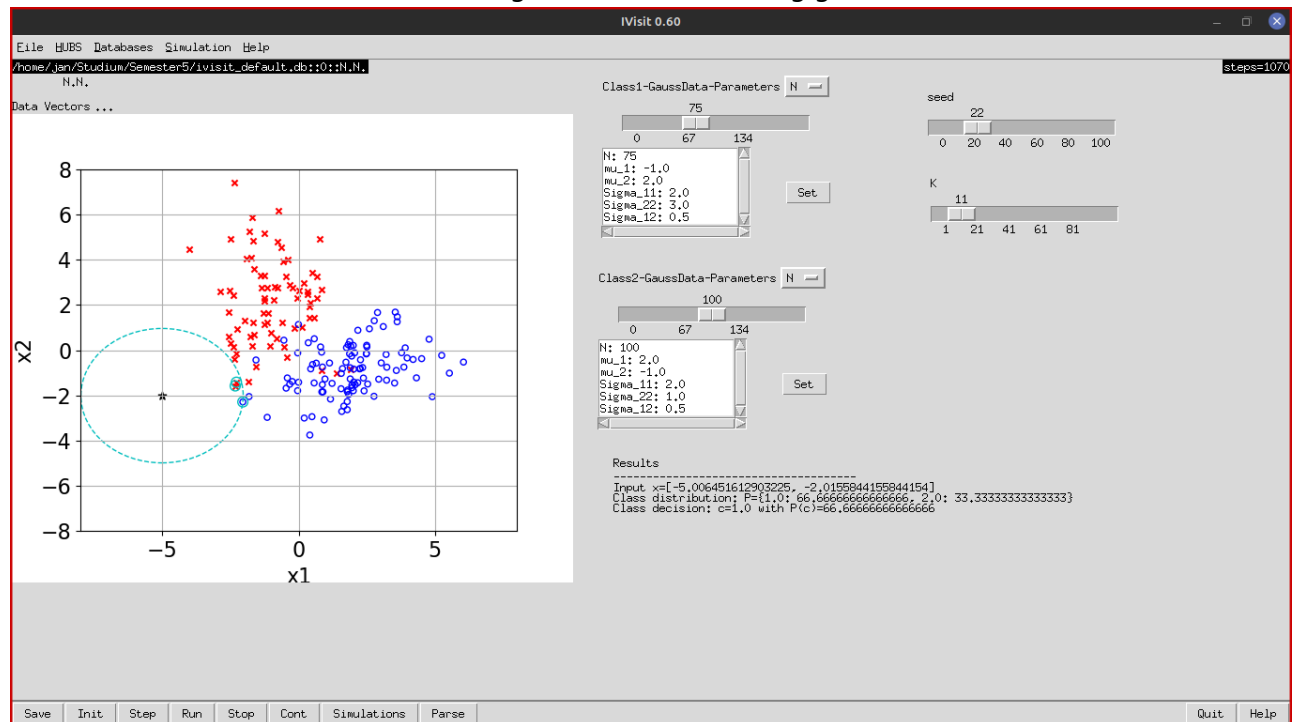
```
(base) jan@jan-P6640:~/Studium/Semester5$ /home/jan/anaconda3/bin/python /home/jan/Studium/Semester5/ILS/ILS_Pr
aktikum/Aufgabe1/versuch1_V2/V1A2/KNearestNeighborSearch.py
Data matrix X=
[[ 1  2  3]
 [-2  3  4]
 [ 3 -4  5]
 [ 4  5 -6]
 [-5  6  7]
 [ 6 -7  8]]
Class labels t= [0 1 2 0 1 2]
Test vector x= [ 3.5 -4.4  5.3]
Euklidean distances to x: [7.24568837309472, 9.311283477587825, 0.7071067811865476, 14.707141122597553, 13.538
833036861043, 4.505552130427524]
idx_KNN= [2 5 0]
The K Nearest Neighbors of x are the following vectors:
The 1 th nearest neighbor is: X[ 2 ]= [ 3 -4  5] with distance 0.7071067811865476 and class label 2
The 2 th nearest neighbor is: X[ 5 ]= [ 6 -7  8] with distance 4.505552130427524 and class label 2
The 3 th nearest neighbor is: X[ 0 ]= [ 1  2  3] with distance 7.24568837309472 and class label 0
Class distribution P= {0: 33.33333333333333, 1: 33.33333333333333, 2: 33.33333333333333}
Most likely class: c= 0 with P(c)= 33.33333333333333
(base) jan@jan-P6640:~/Studium/Semester5$
```

e)

- Die Funktion `def onPressedB1_datvec(self,event)` ist dafür da, um mit der Maus im Graphen eine (x,y) Koordinate auszuwählen, welche dann als neuer Datenvektor x verwendet werden soll. Die Funktion holt sich die x und die y Koordinaten, die mit der Maus ausgewählt wurden.
- Die Funktion `def bind(self,parent=None,display=None)` ist dafür zuständig das Bild bzw den Graphen an das Mausclick Event zu binden. Das Event soll nämlich nur ausgelöst werden wenn in einer Region auf dem Graphen geklickt wird.
- Der türkise Kreis um den Datenvektor x wird sehr groß, wenn x sehr weit entfernt von den Trainingsdaten X ist.
- Falls ein sehr großes K gewählt wird, bedeutet dass, das ein sehr großer Teil der X Daten mit einbezogen werden müssen. Somit ist dann auch die Region die betrachtet werden muss sehr groß und

die Klassenentscheidung für den neuen Datenvektor x wird ungenauer.

- Der Vorteil eines ungeradem K ist, dass niemals alle Klassen mit der gleichen Verteilung vorkommen können. Somit kann immer eine eindeutige Klassenentscheidung getroffen werden.



f)

Die Naive brechnung bei KNN werden $O(ND)$ Schritte benötigt also Linear. Um das Verfahren zu beschleunigen, können KD-Trees verwendet werden, die bei guter Pivotisierung eine logarithmische Laufzeit erreichen, also $O(\log N)$.

Aufgabe 3

a)

Der Abstrakte Konstruktor der Klasse Classifier, übergibt den Paramter C an sein Attribut C. C legt die anzahl an möglichen Klassen fest, in die die Daten unterteilt werden können.

Die Methode fit wird für das Trainieren des Klassifikators benutzt. Die Methode bekommt die Parameter X und T, wobei X die Daten und T die dazugehörigen Klassenlabel sind.

Die Methode predict Klassifiziert anhand des Trainings neu übergabene Daten ein. Als Parameter übergibt man Predict z.B. einen neuen Vektor um ihn Klassifizieren zu lasse.

Die Methode Crossvalidate teilt die Daten in S Teile auf. Der Parameter X ist für die Übergabe der Daten zuständig und der Parameter S gibt an, in wie viele Blöcke die Daten X geteilt werden sollen. Der dritte Parameter T sind die Klassenlabel für die Daten in X. Die Methode Crossvalidate macht eine Kreuzvalidierung mit dem Entsprechenden Modell in dem sie implementiert wird. Dies bedeutet, dass die Daten in X in S Blöcke geteilt werden und ein Block nicht zum Trainieren verwendet wird, sondern zum Validieren. Die Kreuzvalidierung macht jedoch mit jedem Block aus X eine Validierung und mittelt daraus dann die Genauigkeit.

b)

- Das Array in idxS enthält die Einträge bei den Parametern N=9 und S=3 [range(0, 3), range(3, 6), range(6, 9)]. Das Array aus 9 Daten soll in 3 gleich große Teile aufgeteilt werden. somit kommen dann die drei ranges zu stande.
- idxVal enthält für jeden Schleifendurchgang einen der range Einträge aus der Liste idxS. idxTrain enthält die restlichen Blöcke die nicht in der Range von idxVal liegen.
- In perm ist eine zufällige Reihenfolge aus Indexen für X. Dies ist sinnvoll, da wir die Daten in X nicht in der vorgegeben Reihenfolge verarbeiten wollen, da diese schon eine Art Ordnung haben könnten. Somit indizieren wir mit einer Range aus den perm Indexen auf die Daten X und die Label T mit denen wir dann das Modell trainieren wollen.
- 1 Durchlauf: x[3], x[1], x[0] 2 Durchlauf: x[8], x[2], x[4] 3 Durchlauf: x[7], x[5], x[6]
- Der Returnwert err ist der prediction error, die Formel dafür befindet sich auf Seite 115 Nummer 2.108 $(FP + FN) / (TP + FP + FN + TN)$. Der Returnwert MatCp ist die Confusion Matrix welche die Werte False Positive (FP), False Negative (FN), True Positive (TP) und True Negative (TN) enthält.

f)

```

• (base) jan@jan-P6640:~/Studium/Semester5$ /home/jan/anaconda3/bin/python /home/jan/Studium/Semester5/ILS/ILS_Pr
aktikum/Aufgabe1/versuch1_V2/V1A3/Classifier.py
/home/jan/anaconda3/lib/python3.9/site-packages/scipy/__init__.py:155: UserWarning: A NumPy version >=1.18.5 an
d <1.25.0 is required for this version of SciPy (detected version 1.26.1
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}")
Data matrix X=
[[ 1  2  3]
 [-2  3  4]
 [ 3 -4  5]
 [ 4  5 -6]
 [-5  6  7]
 [ 6 -7  8]]
Class labels T= [0 1 2 0 1 2]
Test vector x= [ 3.5 -4.4  5.3]
Euklidean distances to x: [7.24568837309472, 9.311283477587825, 0.7071067811865476, 14.707141122597553, 13.538
833036861043, 4.505552130427524]

Classification with the naive KNN-classifier:
Test vector is most likely from class y_hat= 2
A-Posteriori Class Distribution: prob(x is from class i)= 0.6666666666666666
Indexes of the K= 3 nearest neighbors: idx_knn= [2 5 0]

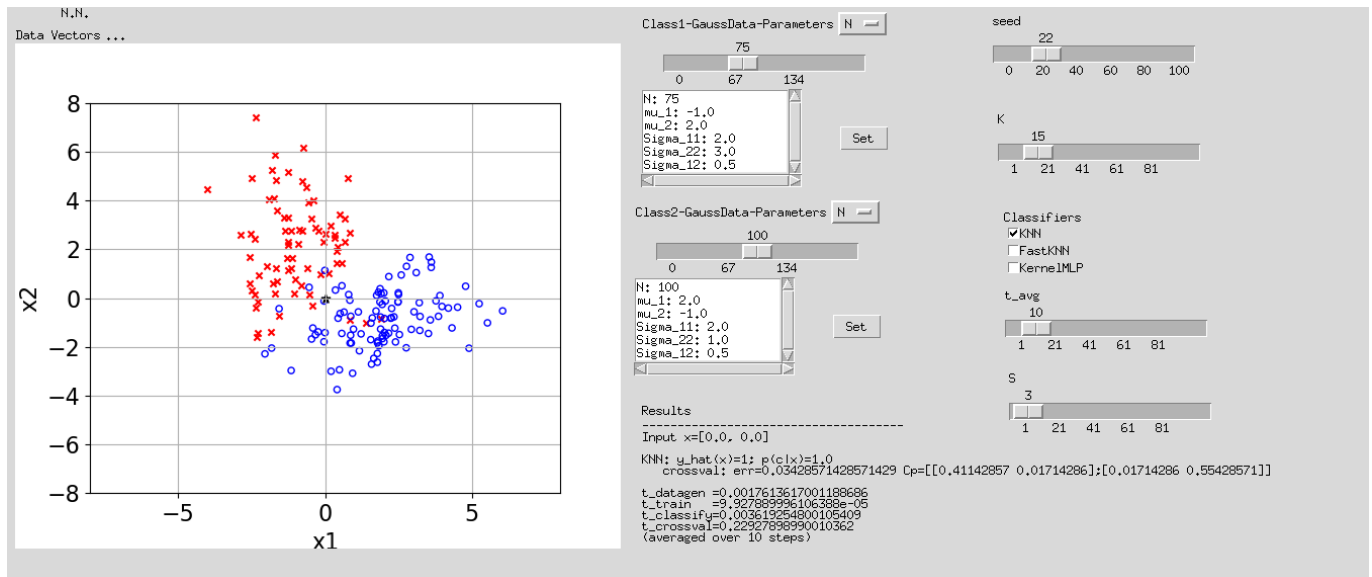
Classification with the fast KNN-classifier based on kd-trees:
Test vector is most likely from class y_hat= 2
A-Posteriori Class Distribution: prob(x is from class i)= 0.6666666666666666
Indexes of the K= 3 nearest neighbors: idx_knn= [2 5 0]


Classification with the Kernel-MLP:
Test vector is most likely from class y_hat= 2
Model outputs y= [ 3.57588397e-06 -4.88406252e-08 1.00029482e+00]

CrossValidation with S= 2 for KNN-Classifer:
err= 0.5
matCp= [[0.16666667 0.          0.16666667]
 [0.          0.33333333 0.33333333]
 [0.          0.          0.          ]]
• (base) jan@jan-P6640:~/Studium/Semester5$

```

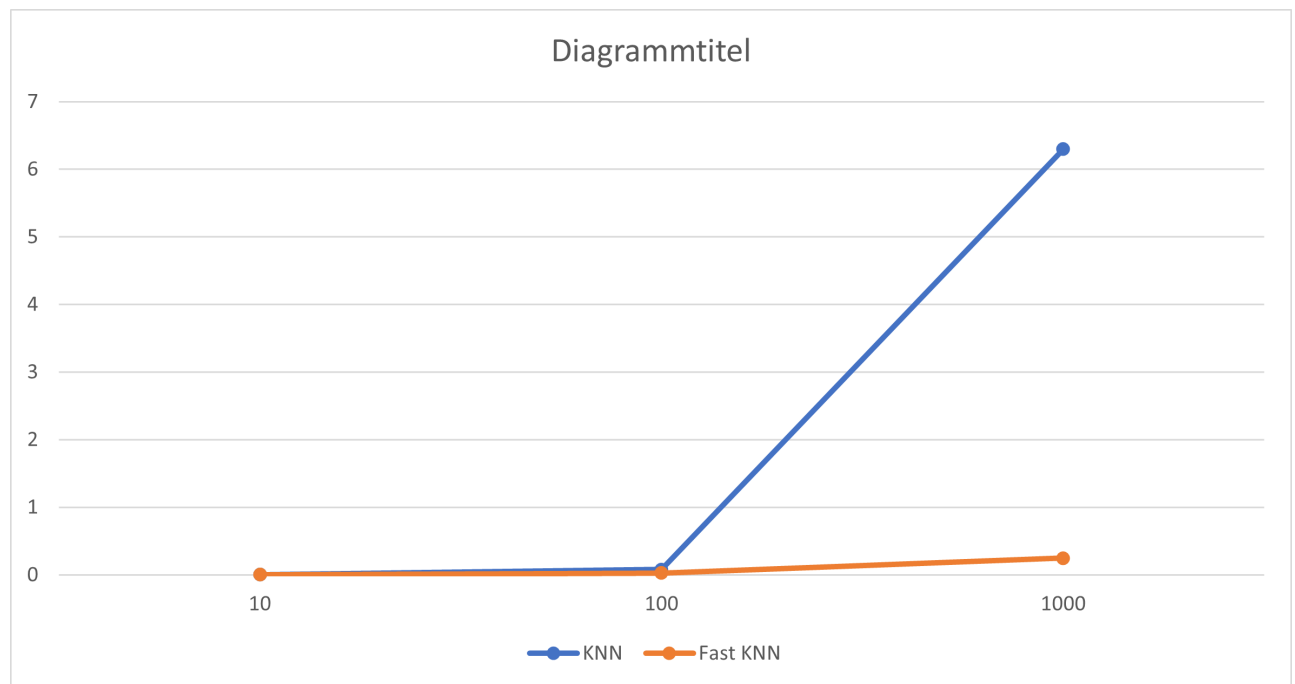
g)



Der kleinste Error, wurde mit $K = 15$ erreicht.  Aufgabe 3g MLP

- Was passiert im Fall vieler Daten (N_1, N_2 groß), falls die Daten gut getrennt sind gegenüber dem Fall wenn die Daten stark überlappen?
 - Bei vielen Daten kann die Invertierung der Gram-matrix sehr aufwändig werden. $O(n^3)$ Rechenschritte.
- Warum funktioniert KernelMLP im letzteren Fall nicht so gut?
 - Wenn die Daten stark überlappen, wird die Aufgabe für das Modell schwieriger. In diesem Fall kann es schwieriger sein, klare Trennlinien zwischen den Klassen zu ziehen. KernelMLP kann in diesem Szenario möglicherweise nicht so gut funktionieren, weil es möglicherweise Schwierigkeiten hat, die komplexen nicht-linearen Muster zu erfassen, die durch die Überlappung entstehen. Modelle mit nicht-linearen Kernen können in solchen Fällen anfälliger für Overfitting sein.
- Vergleichen Sie für die Datenparameter von Aufg.1c die Laufzeiten der drei Verfahren für Kreuzvalidierung für Datenzahl $N = 10, 100, 1000$ (mit $N_1 = N_2 = N/2$). Tragen Sie die Werte jeweils in

ein Schaubild ein.



Aufgabe 4

b)

```
-2.317e+01 -2.200e-01 -4.220e+00]
[ 4.000e+01  3.900e+01  5.800e+01  8.200e+01  6.100e+01  9.900e+01
 8.900e+01  2.600e+01  5.700e+01  7.399e+01  1.291e+01  4.192e+01
 1.733e+01 -3.482e+01 -3.619e+01 -1.107e+01  4.280e+00 -1.900e-01
-1.872e+01 -2.610e+00 -8.380e+00 -2.056e+01 -1.510e+00 -6.680e+00
-2.116e+01 -3.420e+00 -6.610e+00]
[ 5.300e+01  2.700e+01  4.900e+01  9.500e+01  4.900e+01  9.200e+01
 6.300e+01  2.500e+01  5.400e+01  6.697e+01  2.443e+01  4.928e+01
 8.080e+00 -2.253e+01 -2.825e+01  1.978e+01  3.750e+00  9.200e-01
-2.565e+01 -2.090e+00 -5.950e+00 -3.927e+01 -2.130e+00 -8.730e+00
-3.073e+01 -2.420e+00 -5.580e+00]
[ 5.100e+01  5.700e+01  7.700e+01  9.000e+01  8.900e+01  1.230e+02
 9.700e+01  4.700e+01  8.300e+01  6.491e+01 -5.210e+00  2.145e+01
 1.221e+01 -6.290e+01 -6.040e+01 -1.675e+01 -1.685e+01 -2.644e+01
-2.097e+01 -1.760e+00 -5.050e+00 -2.201e+01 -9.300e-01 -5.600e+00
-2.226e+01 -3.280e+00 -6.390e+00]
T_txt[0..9]=
['d' 'h' 's' 's' 'd' 'h' 's' 'd' 's' 'o ']
T[0..9]=
[2 1 0 0 2 1 0 2 0 3]

S= 5 fold cross validation using the 3 -NNClassifier with KD-Trees yields the following results:
Classification error probability err = 0.060606060606061
Accuracy = 0.9393939393939394
Confusion Probabilities matrix Cp[i,j]=p(true class i,predicted class j) =
[[0.29292929 0. 0. 0. ]
[0. 0.2020202 0. 0. ]
[0. 0. 0.22727273 0. ]
[0. 0.02020202 0.04040404 0.21717172]]
Computing time = 0.05219371599923761 sec

accuracy=
[[0.93939394]]

p_classerror=
[[0.06060606]]

minimal err= 0.060606060606061 for S= 5 K= 3
(base) jan@jan-P6640:~/Studium/Semester5$
```

c)

Für S=10 und K=9 hatten wir einen Error von 0. Das zweit beste Ergebniss wurde mit S=10 und K=1 erreicht, hier beträgt der Error nur 0,005.

```

Confusion Probabilities matrix Cp[i,j]=p(true class i,predicted class j) =
[[0.2020202 0. 0. 0. ]
 [0. 0.24242424 0. 0. ]
 [0. 0. 0.2020202 0. ]
 [0. 0. 0.01010101 0.34343434]]
Computing time = 0.05842948499957856 sec

S= 10 fold cross validation using the 5 -NNClassifier with KD-Trees yields the following results:
Classification error probability err = 0.10101010101010101
Accuracy = 0.8989898989898989
Confusion Probabilities matrix Cp[i,j]=p(true class i,predicted class j) =
[[0.2020202 0.05050505 0. 0. ]
 [0. 0.19191919 0. 0. ]
 [0. 0. 0.4040404 0. ]
 [0. 0. 0.05050505 0.1010101 ]]
Computing time = 0.0568348730012076 sec

S= 10 fold cross validation using the 7 -NNClassifier with KD-Trees yields the following results:
Classification error probability err = 0.10101010101010101
Accuracy = 0.8989898989898989
Confusion Probabilities matrix Cp[i,j]=p(true class i,predicted class j) =
[[0.1010101 0. 0. 0. ]
 [0. 0.2020202 0. 0. ]
 [0. 0. 0.4040404 0. ]
 [0. 0. 0.1010101 0.19191919]]
Computing time = 0.05934544000047026 sec

S= 10 fold cross validation using the 9 -NNClassifier with KD-Trees yields the following results:
Classification error probability err = 0.0
Accuracy = 1.0
Confusion Probabilities matrix Cp[i,j]=p(true class i,predicted class j) =
[[0.29292929 0. 0. 0. ]
 [0. 0.15151515 0. 0. ]
 [0. 0. 0.4040404 0. ]
 [0. 0. 0. 0.15151515]]
Computing time = 0.058700067000245326 sec

S= 10 fold cross validation using the 11 -NNClassifier with KD-Trees yields the following results:
Classification error probability err = 0.06060606060606061

```

d)

- beim Verwenden von $h_z = \tanh$ geht etwas schief. Warum? Was kann man dagegen tun?
 - \tanh hat den Sättigungsbereich bei 1 und -1. Wenn die Dentratischen Potentiale a große Werte annehmen, dann ist man im Sättigungsbereich der \tanh Funktion, welche dann immer 1 oder -1 aus Funktionswert ausgibt. In der Ergebnismatrix sind dann nur gleiche Vektoren zu finden. Sobald in mehreren Zeilen eine 1 zu finden ist, ist die Matrix Linear abhängig und kann nicht mehr Invertiert werden. Um dies zu verhindern können andere Aktivierungsfunktionen verwendet

werden, wie die Kubische Funktion x^3 zum Beispiel.

```

 9.700e+01  4.700e+01  8.300e+01  6.491e+01 -5.210e+00  2.145e+01
 1.221e+01 -6.290e+01 -6.040e+01 -1.675e+01 -1.685e+01 -2.644e+01
-2.097e+01 -1.760e+00 -5.050e+00 -2.201e+01 -9.300e-01 -5.600e+00
-2.226e+01 -3.280e+00 -6.390e+00]]
T_txt[0..9]=
['d' 'h' 's' 's' 'd' 'h' 's' 'd' 's' 'o ']
T[0..9]=
[2 1 0 0 2 1 0 2 0 3]
S= 2 hz= <function hz_sgn_sqrt at 0x7eff34181c10>
S= 2 hz= <function hz_sgn_log at 0x7eff24b91c10>
S= 2 hz= <function hz_sgn_square at 0x7eff22ebbaf0>
S= 2 hz= <function hz_cubic at 0x7eff22ebbb80>
S= 3 hz= <function hz_sgn_sqrt at 0x7eff34181c10>
S= 3 hz= <function hz_sgn_log at 0x7eff24b91c10>
S= 3 hz= <function hz_sgn_square at 0x7eff22ebbaf0>
S= 3 hz= <function hz_cubic at 0x7eff22ebbb80>
S= 5 hz= <function hz_sgn_sqrt at 0x7eff34181c10>
S= 5 hz= <function hz_sgn_log at 0x7eff24b91c10>
S= 5 hz= <function hz_sgn_square at 0x7eff22ebbaf0>
S= 5 hz= <function hz_cubic at 0x7eff22ebbb80>
S= 10 hz= <function hz_sgn_sqrt at 0x7eff34181c10>
S= 10 hz= <function hz_sgn_log at 0x7eff24b91c10>
S= 10 hz= <function hz_sgn_square at 0x7eff22ebbaf0>
S= 10 hz= <function hz_cubic at 0x7eff22ebbb80>

accuracy=
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]

p_classerror=
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]

mininmal err= 0.0101010101010102 for S= 5 hz= <function hz_sgn_square at 0x7eff22ebbaf0>
○ (base) jan@jan-P6640:~/Studium/Semester5$ 

```

e)

f)

Bei KNN wird angenommen das ähnliche Daten im Merkmalsraum nahe beieinander liegen. Wenn dies nicht der Fall ist kann es zu Problemen kommen, da die berechneten Distanzen dann sehr lange sein können. Um dieses Problem zu beheben müssen in diesem Fall die Trainingsdaten Normalisiert werden.