

## SMD-Abgabe

# 5. Übungsblatt

Lars Kolk

[lars.kolk@tu-dortmund.de](mailto:lars.kolk@tu-dortmund.de)

Julia Sobolewski

[julia.sobolewski@tu-dortmund.de](mailto:julia.sobolewski@tu-dortmund.de)

Jannine Salewski

[jannine.salewski@tu-dortmund.de](mailto:jannine.salewski@tu-dortmund.de)

Abgabe: 29.11.2018

15	16	17	18	$\Sigma$
7/8	5/6	25/3	25/4	17/21

TU Dortmund – Fakultät Physik

# 1 Aufgabe 16

## 1.1 a)

Entropie berechnen:

$$\begin{aligned} S &= - \sum_i p_i \log_2(p_i) \\ &= - [P(\text{Fuball} = \text{True}) \log_2(P(\text{Fuball} = \text{True})) + P(\text{Fuball} = \text{False}) \log_2(P(\text{Fuball} = \text{False}))] \\ &= 0,9402 \quad \checkmark \quad 0,5/0,5 \end{aligned}$$

mit

$$\begin{aligned} P(\text{Fuball} = \text{True}) &= \frac{9}{14} \quad ?? \\ P(\text{Fuball} = \text{True}) &= \frac{5}{14} \end{aligned}$$

## 1.2 b)

Informationsgewinn(Wind):

$$\begin{aligned} IG(Y|X) &= H(Y) - H(Y|X) \\ IG(\text{Wind}|Fuball) &= S - \frac{|D_{\text{Wind=True}}|}{|D|} \cdot H(D_{\text{Wind=True}}) - \frac{|D_{\text{Wind=False}}|}{|D|} \cdot H(D_{\text{Wind=False}}) \\ &= 0,04860 \quad \checkmark \quad 2/2 \end{aligned}$$

Hierbei ist  $S$  die in a) berechnete Entropie,  $|D_{\text{Wind=True}}|$  die Anzahl der Wind-Einträge mit "True",  $|D_{\text{Wind=False}}|$  die Anzahl der Wind-Einträge mit "False" und  $|D| = 14$  die Gesamtanzahl der Einträge, mit den bedingten Wahrscheinlichkeiten

$$\begin{aligned} P(\text{Fuball} = \text{True}|\text{Wind} = \text{True}) &= \frac{1}{2} \\ P(\text{Fuball} = \text{False}|\text{Wind} = \text{True}) &= \frac{1}{2} \\ P(\text{Fuball} = \text{True}|\text{Wind} = \text{False}) &= \frac{3}{4} \\ P(\text{Fuball} = \text{False}|\text{Wind} = \text{False}) &= \frac{1}{4} \end{aligned}$$

## 1.3 c)

(Siehe Python-Datei)

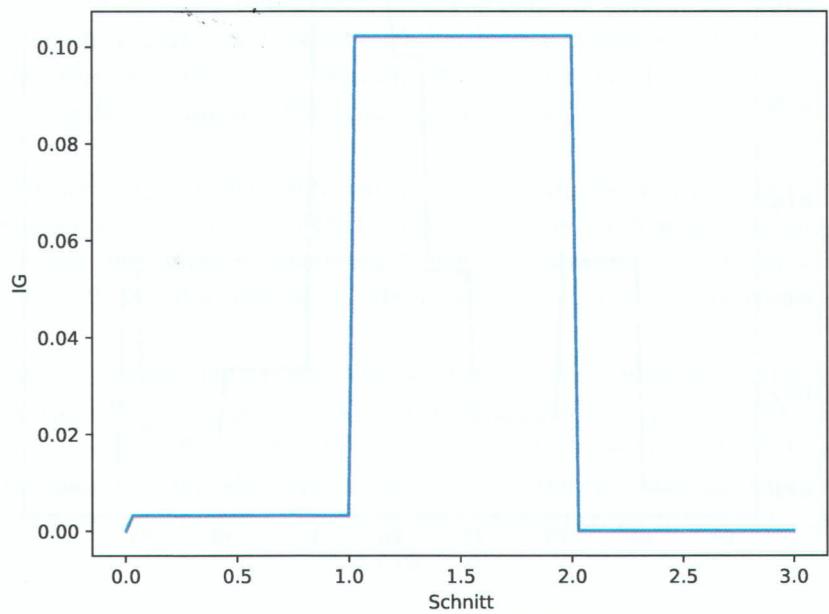


Abbildung 1: Informationsgewinn aus dem Wetters in Abhangigkeit des Schnitts.

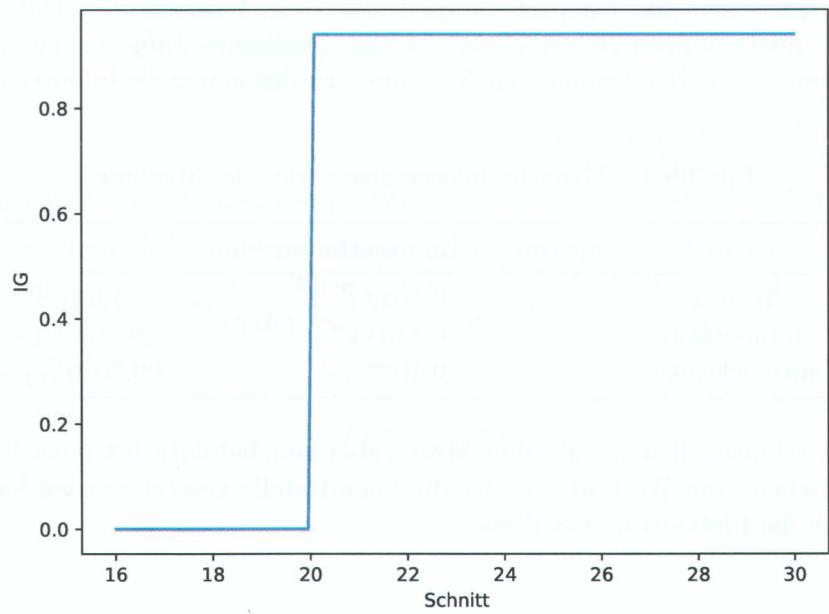
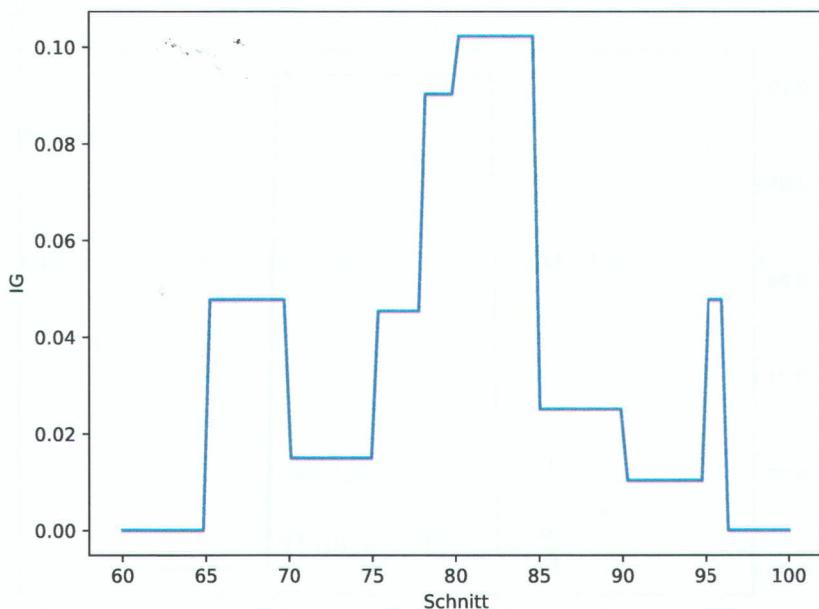


Abbildung 2: Informationsgewinn aus der Temperatur in Abhangigkeit des Schnitts.



**Abbildung 3:** Informationsgewinn aus der Luftfeuchtigkeit in Abhängigkeit des Schnitts.

#### 1.4 d)

Am besten eignet sich das Attribut "Temperatur" zur Trennung der Daten, da hier bei einem Schnitt von etwa  $28,303^{\circ}\text{C}$  ein globaler, maximaler Informationsgewinn von 0,1134 zu erkennen ist. Bei den anderen Attributen ist der maximale Informationsgewinn geringer.

~~Wetter~~ ~~ff~~

c+d  
2,5  
3,5

**Tabelle 1:** Maximaler Informationsgewinn der Attribute

Attribut	maximaler Informationsgewinn	Schnittstelle
Wetter	0,1022 ❌	1,03 ❌ ✓
Temperatur	0,1134 ✓	$28,303^{\circ}\text{C}$ ✓
Luftfeuchtigkeit	0,1022 ✓	80,2020 % ❌

Einteilung in  
① | ②  
→ besser: IG ist  $\approx 0,1134$

Zwar sind die Schnittstellen hier absolute Werte, aber man hat natürlich einen bestimmten Bereich (zwischen zwei Werten), bei dem die Schnittstelle gesetzt werden kann, daher kommen auch die Plateaus in den Plots.

5/6

#### Aufgabe 17

- a) Nicht-numerische Datentypen sollten, wenn möglich (und sinnvoll), in Zahlen ausgedrückt werden um mit ihnen besser arbeiten zu können. Im Fall der Titanic könnte

z.B. der Status der Passagiere (also ob tot oder lebendig) als 0 oder 1 dargestellt werden. Sollte es jedoch nicht-numerische Datentypen geben, die sich nicht sinnvoll in Zahlen ausdrücken lassen, könnte man diese verfallen lassen, sofern sie für die Untersuchung irrelevant sind. Ebenso könnte man dann mit nicht-numerischen Datentypen weiterarbeiten, sofern dies die Aufbereitung des Datensatzes nicht erschwert. ✓ 0,5/0,5

b) Die Normierung der Attribute kann sinnvoll sein, da Attribute bei verschiedenen Größenordnungen verschieden gewichtet sein können. Dadurch könnte einigen Attributen mehr Bedeutung als anderen zugemessen werden, obwohl die Größenordnung nicht ausschlaggebend für die Wichtigkeit ist. Dem kann durch Normierung vorgebeugt werden. ✓ 0,5/0,5

wählen  
oder häufigsten  
Wert  
nehmen

c) Sofern nur wenige Daten des Datensatzes Lücken aufweisen, können diese ggf. verworfen werden. Das ist jedoch auch vom zu untersuchenden Datensatz abhängig, da auch fehlende Informationen Informationen sind. Wenn die betroffenen Daten dennoch in die Untersuchung eingehen oder separat auf Korrelation mit ihren Attributen untersucht werden sollen, kann es sinnvoll sein, die Lücken einheitlich zu füllen (z.B. alles auf NaN setzen). NOCH 0,5/0,5 P Lars einfach nein

d) Die Datensätze sollten vom selben Typ sein. Ebenso könnte es sinnvoll sein, die Datensätze vor dem zusammenfügen entsprechend zu labeln, sodass durch das Zusammenfügen keine Informationen verloren gehen. ✓ 0,5/0,5

e) Redundante Attribute sind vor dem Trainieren der Klassifizierers zu entfernen. Dies kann z.B. per Hand, aber auch durch Entscheidungsbäume geschehen, die einen günstige Schnitt mit einem hohen Informationsgehalt finden. ✓ Simulation? ↓ 2,5/3

18

a) Mit der bedingten Wahrscheinlichkeit folgt:

$$P(F|W) = \frac{P(F \cap W)}{P(W)} = \frac{\frac{P(F \cap W)}{P(F)} \cdot P(F)}{P(W)} = \frac{P(W|F) \cdot P(F)}{P(W)} \quad \checkmark \quad (1)$$

keine Attribute simulieren, die nicht in den Daten enthalten sind 0,5/1

b) Mit

$$P(F) = 9/14 \quad \checkmark \quad (2)$$

folgt mit der Wahrscheinlichkeit der gesuchten Attributen in  $F$

$$P(W|F=j_2)$$

$$P_{Wind_F} = \frac{1}{3} \quad \checkmark \quad (3)$$

$$P_{Temp_F} = \frac{1}{3} \quad \checkmark \quad (4)$$

$$P_{Feucht_F} = \frac{1}{3} \quad \checkmark \quad (5)$$

$$P_{Aussicht_F} = \frac{2}{9} \quad \checkmark \quad (6)$$

und mit der Gesamtwahrscheinlichkeit der Ereignisse

$$P(W|F=\text{nein}) = \frac{3}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{1}{5} = \frac{36}{625} \quad P_{Wind_{Stark}} = \frac{6}{14} \quad (7)$$

$$P(F=\text{nein}) = \frac{5}{14} \quad P_{Temp_{Kalt}} = \frac{6}{14} \quad (8)$$

$$P_{Aussicht_{Sonnig}} = \frac{3}{14} \quad (9)$$

$$P_{Feucht_{Hoch}} = \frac{7}{14} \quad (10)$$

$$N = P(W|F=j_2) \cdot P(F=j_2) + P(W|F=\text{nein}) \cdot P(F=\text{nein}) \quad (11)$$

und dem in der Aufgabenstellung gegebenen Zusammenhang

$$P(F|W) = \prod_i P(x_i|W) \quad (12)$$

der Zusammenhang

$$P(F|W) = \frac{P_{Wind_F} \cdot P_{Temp_F} \cdot P_{Feucht_F} \cdot P_{Aussicht_F} \cdot P(F)}{P_{Wind_{Stark}} \cdot P_{Temp_{Kalt}} \cdot P_{Aussicht_{Sonnig}} \cdot P_{Feucht_{Hoch}}} \quad (13)$$

$$P(F|W) = 26,9\%$$

$$P(F=j_2|W) = \frac{P(W|F=j_2) \cdot P(F=j_2)}{N}$$

*Normierung*  
*Hilflich*

- c) Die Wahrscheinlichkeit beträgt 0, da der Datensatz keine Aufzeichnungen von stattgefundenen Spielen bei heißen Temperaturen enthält. Dies erzeugt einen Faktor 0, der das gesamte Produkt 0 werden lässt. Ein größerer Datensatz könnte das Problem beheben.

*Was macht man in diesem Fall?*

$$0,5/1$$

- Laplace-Estimation : +1

(Attribut weglassen)

$$25/4$$

## Aufgabe 15: $k$ -NN Klassifikation

a)

Unterscheiden sich die Attribute stark in ihrer Größenordnung, dann werden diese unterschiedlich stark gewichtet wodurch der Klassifizierer nicht mehr so genau arbeiten kann.

Lösung?

→ Normierung

1 P.

b)

Der k-NN wird als "lazy learner" bezeichnet, weil der Algorithmus die Trainingsdaten nicht verwendet, um zu generalisieren, sondern den Trainingsdatensatz auswendig lernt.

0.5 P.

Anwendungsphase relativ lang. Trainingsphase sehr kurz, weil Datensatz einfach auswendig gelernt wird.

c)

```

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)

            sorted_distance_list = distance_list[np.argsort(distance_list[:,0])]
            neighbours = sorted_distance_list[:self.k]

            for j in range(self.k):
                if

```

d)

```
In [2]: import pandas as pd
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
```

```
In [27]: def getdata(file, key, log = False):
    data = pd.read_hdf(file, key)

    hits = np.array(data.NumberOfHits[~np.isnan(data.NumberOfHits)])
    if log == True:
        hits = np.log10(hits)

    x = np.array(data.x[~np.isnan(data.x)])
    y = np.array(data.y[~np.isnan(data.y)])

    return np.matrix([hits, x, y]).T

def splitdata(signal, background, n_train, n_sig, n_back, shuffle = False, seed = 0):
    if shuffle == True:
        signal, background = shuffle(signal, background, random_state=seed)

    signal = signal[:n_train+n_sig]
    background = background[:n_train+n_back]

    train_sig, test_sig = np.split(signal, [n_train])
    train_back, test_back = np.split(background, [n_train])

    train_data = np.concatenate((train_back, train_sig))

    labels = np.concatenate((np.zeros(n_train), np.ones(n_train))) # 0=Background
    , 1=Signal

    return train_data, test_back, test_sig, labels
```

```
In [14]: signal = getdata('NeutrinoMC.hdf5', 'Signal')
background = getdata('NeutrinoMC.hdf5', 'Background')

n_train = 5000
n_sig = 10000
n_back = 20000
k = 10
```

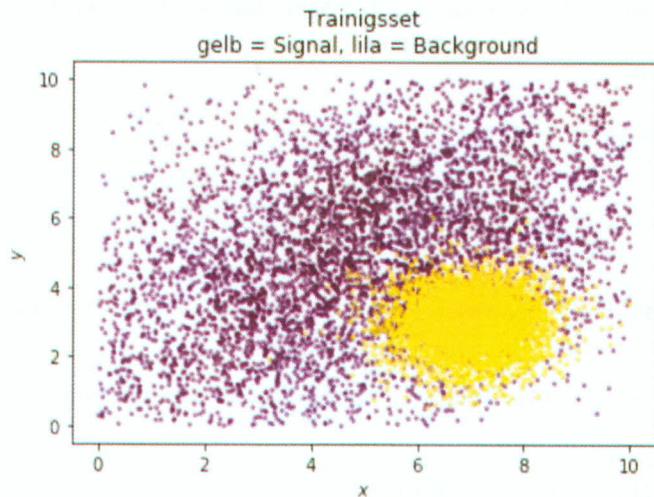
```
In [28]: train_data, test_back, test_sig, labels = splitdata(signal, background, n_train,
n_sig, n_back)
```

```
In [18]: x = np.ravel(train_data[:,1])
y = np.ravel(train_data[:,2])

plt.scatter(x, y, marker='.', c = labels, alpha = 0.5,
           s=15) #, edgecolor='k', linewidths=0.3)

plt.title('Trainigset \n gelb = Signal, lila = Background')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
```

Out[18]: Text(0,0.5,'\$y\$')



```
In [23]: knn = KNN(k)

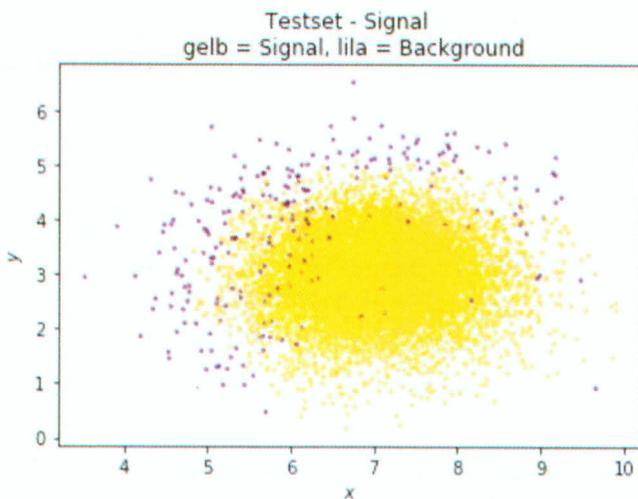
x_sig = np.ravel(test_sig[:,1])
y_sig = np.ravel(test_sig[:,2])

knn.fit(train_data, labels)
predict_labels_sig = knn.predict(test_sig)

plt.scatter(x_sig, y_sig, marker='.', c = predict_labels_sig, alpha = 0.5,
           s=15) #, edgecolor='k', linewidths=0.3)

plt.title('Testset - Signal \n gelb = Signal, lila = Background')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
```

Out[23]: Text(0,0.5,'\$y\$')



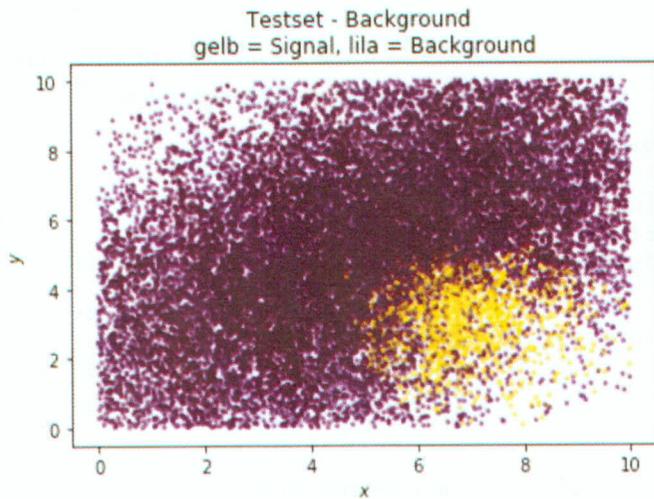
```
In [24]: x_back = np.ravel(test_back[:,1])
y_back = np.ravel(test_back[:,2])

knn.fit(train_data, labels)
predict_labels_back = knn.predict(test_back)

plt.scatter(x_back, y_back, marker='.', c = predict_labels_back, alpha = 0.5,
           s=15) #, edgecolor='k', linewidths=0.3)

plt.title('Testset - Background \n gelb = Signal, lila = Background')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
```

Out [24]: Text(0,0.5,'\$y\$')

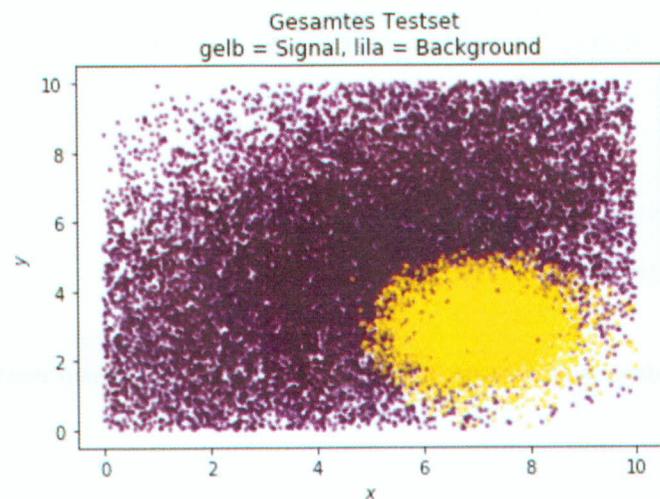


```
In [26]: plt.scatter(x_back, y_back, marker='.', c = predict_labels_back, alpha = 0.5,
                   s=15) #, edgecolor='k', linewidths=0.3)

plt.scatter(x_sig, y_sig, marker='.', c = predict_labels_sig, alpha = 0.5,
           s=15) #, edgecolor='k', linewidths=0.3)

plt.title('Gesamtes Testset \n gelb = Signal, lila = Background')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$')
```

Out [26]: Text(0,0.5,'\$y\$')



```
In [9]: def validation(k, train_data, labels, test_sig, test_back):
    knn = KNN(k)
    knn.fit(train_data, labels)

    predict_sig = np.array(knn.predict(test_sig))
    predict_back = np.array(knn.predict(test_back))

    tp = np.count_nonzero(predict_sig == 1)
    fn = np.count_nonzero(predict_sig == 0)
    fp = np.count_nonzero(predict_back == 1)
    tn = np.count_nonzero(predict_back == 0)

    print('tp =', tp)
    print('fn =', fn)
    print('fp =', fp)
    print('tn =', tn)

    Reinheit = tp/(tp+fp)
    Effizienz = tp/(tp+fn)
    Signifikanz = tp/(np.sqrt(tp+fp))

    print('Reinheit =', Reinheit)
    print('Effizienz =', Effizienz)
    print('Signifikanz =', Signifikanz)
```

```
In [10]: validation(k, train_data, labels, test_sig, test_back)
```

```
tp = 9679
fn = 321
fp = 1824
tn = 18176
Reinheit = 0.8414326697383291 ✓
Effizienz = 0.9679 ✓
Signifikanz = 90.24537002194234 ✓
```

1P.

e)

```
In [29]: log_signal = getdata('NeutrinoMC.hdf5', 'Signal', log=True)
log_background = getdata('NeutrinoMC.hdf5', 'Background', log=True)

train_data, test_back, test_sig, labels = splitdata(log_signal, log_background,
n_train, n_sig, n_back)

validation(k, train_data, labels, test_sig, test_back)

tp = 9734
fn = 266
fp = 1299
tn = 18701
Reinheit = 0.8822623039970996 ✓
Effizienz = 0.9734 ✓
Signifikanz = 92.67114581738896 ✓
```

0,5P.

Werden die  $\log_{10}$ (Hits) statt der Hits verwendet, dann verbessern sich Reinheit, Effizienz und Signifikanz des knn-Algorithmus um 1% bis 4%.

f)

```
In [30]: k = 20

train_data, test_back, test_sig, labels = splitdata(signal, background, n_train,
n_sig, n_back)

validation(k, train_data, labels, test_sig, test_back)

tp = 9656
fn = 344
fp = 2054
tn = 17946
Reinheit = 0.8245943637916311 ✓
Effizienz = 0.9656 ✓
Signifikanz = 89.23162655007467 ✓
```

Bei der Einbeziehung der 20 nächsten Nachbarn für den nicht logarithmierten Datensatz verschlechtern sich die drei Werte, um 1% bis 2%. Am wenigsten ist dabei die Effizienz betroffen.

```
In [31]: train_data, test_back, test_sig, labels = splitdata(log_signal, log_background,
n_train, n_sig, n_back)

validation(k, train_data, labels, test_sig, test_back)

tp = 9809
fn = 191
fp = 1465
tn = 18535
Reinheit = 0.8700549937910236 ✓
Effizienz = 0.9809 ✓
Signifikanz = 92.38165096000478 ✓
```

Beim logarithmierten Datensatz, werden Reinheit und Signifikanz minimal schlechter, während sich die Effizienz verbessert.

0,5P.

7P.  
8P.

# Code fuer Blatt06

Kolk, Sobolewski, Salewski

30. November 2018

```
..../B/7/Blatt06_Kolk_Sobolewski_Salewski/Aufgabe16.py

1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4 from scipy.optimize import curve_fit
5 import uncertainties.unumpy as unp
6 from uncertainties import ufloat
7 from scipy.stats import stats
8
9 Temperatur, Wetter, Luftfeuchtigkeit, Wind, Fußball = np.genfromtxt('Tabelle.txt', unpack=True)
10
11
12 def get_Entropie(p_1, p_2):
13     if p_1==0:
14         S = -p_2*np.log2(p_2)
15     elif p_2==0:
16         S = -p_1*np.log2(p_1)
17     else:
18         S = -(p_1*np.log2(p_1) + p_2*np.log2(p_2))
19     return S
20
21
22 # Entropie berechnen:
23 p_Fußball_False = len(Fußball[Fußball==False])/len(Fußball)
24 p_Fußball_True = 1- p_Fußball_False
25 S = get_Entropie(p_Fußball_False, p_Fußball_True)
26
27
28 def get_IG(D, Schnitt):
29     IG = np.zeros(len(Schnitt))
30
31
32     for l in range(len(Schnitt)):
33         D_u = len(D[D < Schnitt[l]]) # Anzahl der Werte unter dem Schnitt
34         D_ü = len(D)-D_u # Anzahl der Werte über dem Schnitt
35
36
37         if D_u==0 or D_ü==len(D):
38             IG[l]=0 # Falls alle Werte unter oder über dem Schnitt sind, ist
39             # der Informationsgewinn 0
40         else:
41             # Berechnung der bedingten Wahrscheinlichkeiten p_unterSchnitt:
42             Anzahl_unterSchnitt_False = 0
43
44
45             for i in range(len(D)):
46                 if D[i]<Schnitt[l] and Fußball[i]==False:
47                     Anzahl_unterSchnitt_False = Anzahl_unterSchnitt_False+1
48
49
50             p_unterSchnitt_False = Anzahl_unterSchnitt_False/D_u
51             p_unterSchnitt_True = 1-p_unterSchnitt_False
52             S_1 = get_Entropie(p_unterSchnitt_False, p_unterSchnitt_True)
53
54             # Berechnung der Wahrscheinlichkeit p_überSchnitt:
55             Anzahl_überSchnitt_False = 0
```

```

58     for i in range(len(D)):
59         if D[i]>=Schnitt[1] and Fußball[i]==False:
60             Anzahl_überSchnitt_False = Anzahl_überSchnitt_False+1
61
62             p_überSchnitt_False=Anzahl_überSchnitt_False/D_ü
63             p_überSchnitt_True = 1-p_überSchnitt_False
64             S_2 = get_Entropie(p_überSchnitt_False, p_überSchnitt_True)
65
66             IG[1] = S - D_u/len(D)*S_1 - D_ü/len(D)*S_2
67
68     return IG
69
70
71 # IG Wetter
72 Schnitt_Wetter = np.array([0,1,2,3])
73 Schnitt_Wetter = np.linspace(0, 3, 100)
74 IG_Wetter = get_IG(Wetter, Schnitt_Wetter)
75 print('WETTER:')
76 print('Bester Schnitt: ', Schnitt_Wetter[np.argmax(IG_Wetter)])
77 print('Bester IG: ', np.max(IG_Wetter))
78
79 # IG Temperatur:
80 Schnitt_Temperatur = np.linspace(16, 30, 100)
81 IG_Temperatur = get_IG(Temperatur, Schnitt_Temperatur)
82 print('TEMPERATUR:')
83 print('Bester Schnitt: ', Schnitt_Temperatur[np.argmax(IG_Temperatur)])
84 print('Bester IG: ', np.max(IG_Temperatur))
85
86 # IG Luftfeuchtigkeit
87 Schnitt_Luft = np.linspace(60, 100, 100)
88 IG_Luft = get_IG(Luftfeuchtigkeit, Schnitt_Luft)
89 print('LUFTFEUCHTIGKEIT:')
90 print('Bester Schnitt: ', Schnitt_Luft[np.argmax(IG_Luft)])
91 print('Bester IG: ', np.max(IG_Luft))
92
93 # Plot: Wetter
94 plt.plot(Schnitt_Wetter, IG_Wetter)
95 plt.xlabel('Schnitt')
96 plt.ylabel('IG')
97 plt.tight_layout()
98 plt.savefig('IG_Wetter.pdf')
99 plt.clf()
100
101 # Plot: Temperatur
102 plt.plot(Schnitt_Temperatur, IG_Temperatur)
103 plt.xlabel('Schnitt')
104 plt.ylabel('IG')
105 plt.tight_layout()
106 plt.savefig('IG_Temperatur.pdf')
107 plt.clf()
108
109 # Plot: Luftfeuchtigkeit
110 plt.plot(Schnitt_Luft, IG_Luft)
111 plt.xlabel('Schnitt')
112 plt.ylabel('IG')
113 plt.tight_layout()
114 plt.savefig('IG_Luft.pdf')
115 plt.clf()

```

.../B/7/Blatt06\_Kolk\_Sobolewski\_Salewski/Aufgabe18.py

```

1 #Wahrscheinlichkeit für ein Spiel:
2 P_F=9/14
3 #Wahrscheinlichkeiten für die geg. Parameter:
4 P_Wind_Stark=6/14
5 P_Temp_Kalt=6/14
6 P_Aussicht_Sonnig=3/14
7 P_Feucht_Hoch=7/14
8 #Wahrscheinlichkeit für die geg. Parameter in F:
9 P_Wind_F=1/3
10 P_Temp_F=1/3

```

```
11 P_Feucht_F=1/3
12 P_Aussicht_F=2/9
13 #Gesamtwahrscheinlichkeit:
14 P_F_W=(P_Wind_F*p_Temp_F*p_Feucht_F*p_Aussicht_F)*p_F/(P_Wind_Stark*p_Temp_Kalt*p_Aussicht_Sonnig*p_Feucht_Ho
15 print(P_F_W)
```