

# SMD-Übungsblatt 6

15	16	17	18	Z
4/8	4,5/6	0/3	4/4	12,5/21

Abgabe: 29.11.18

Yvonne Kasper yvonne.kasper@udo.edu ,  
Robert Appel robert.appel@udo.edu ,  
Julian Schröer julian.schroeer@udo.edu

## 1 Aufgabe1

a) Sind die Populationen unterschiedlich stark vertreten neigt der KNN Algorithmus dazu die Elemente eher in die stärker vertretene Population einzuordnen, da es immer mehr k-Nachbarn von der größeren Population geben wird.

OP.

b) Der Algorithmus wird als „lazy learner“ bezeichnet, da ein Teil des Lehrprozesses beinhaltet, dass der Trainingsdatensatz teilweise abgespeichert wird. Das wird in der Literatur als „lazy learning“ bezeichnet. Im Vergleich zu anderen Algorithmen sehr kosten günstig, da er nur auf jedes Element des zu analysierenden Datensatzes angewendet werden muss (N Iterationen). Dabei spielt es keine Rolle in wieviel Dimensionen das Problem analysiert werden muss, so lange es ein geeignetes Abstandsmaß gibt. Genau an dieser Stelle sticht der KNN den SVM aus, da die Recheneffizienz des SVM stark von der Dimension des Problems abhängt.

OR.

c)

```
1 #### one classy boi
2 class KNN:
3     '''KNN Classifier.
4
5     Attributes
6
7     k : int
8         Number of neighbors to consider.
9     ...
10    def __init__(self, k):
11        '''Initialization.
12        Parameters are stored as member variables/attributes.
13
14    Parameters
15
16    k : int
17        Number of neighbors to consider.
18    ...
19    self.k = k
20    #Hier fehlt glaube ich noch ein self.traindata und self.trainlabels
21    def fit(self, X, y):
22        '''Fit routine.
23        Training data is stored within object.
24
25    Parameters
26
27    X : numpy.array, shape=(n_samples, n_attributes)
28        Training data.
29    y : numpy.array shape=(n_samples)
```

```

30     Training labels .
31     ...
32 # Code
33     self.traindata = X
34     self.trainlabels = y
35
36 def predict(self , X):
37     '''Prediction routine .
38     Predict class association of each sample of X.
39
40 Parameters
41
42 X : numpy.array , shape=(n_samples , n_attributes)
43     Data to classify .
44
45 Returns
46
47 prediction : numpy.array , shape=(n_samples)
48     Predictions , containing the predicted label of each sample.
49     ...
50 prediction = np.array []
51 # Code
52 for x in X:
53     abst1 = self.traindata.T[0] - x.T[0]
54     abst2 = self.traindata.T[1] - x.T[1]
55     abst3 = self.traindata.T[2] - x.T[2]
56     abst = np.sqrt(abst1**2 + abst2**2 + abst3**2)
57     indis = np.argsort(abst)
58     indis = indis [:self.k]
59     labels = np.array (self.trainlabels [indis])
60     u, inds = np.unique (labels , return_inverse=True)
61     label = u[np.argmax(np.bincount(inds))]
62     prediction = np.append(prediction , label)
63 return prediction

```

plots/class\_structure.py

3SP.

d,e,f) Werte einlesen, Trainingsdatenset erstellen, Datenset erstellen:

```

1 ###read in some data
2 Background = pd.read_hdf('NeutrinoMC.hdf5' , key= 'Background')
3 Signal = pd.read_hdf('NeutrinoMC.hdf5' , key= 'Signal')
4
5 ### Some training data
6 TrainingsetX = np.ndarray(shape=(5000,3)) 5000 5000
7 TrainingsetX.T[0] = np.append(Signal.x[:1666] , Background.x[:3334])
8 TrainingsetX.T[1] = np.append(Signal.y[:1666] , Background.y[:3334])
9 TrainingsetX.T[2] = np.append(Signal.NumberOfHits[:1666] , Background.NumberOfHits[:3334])
10 TrainingsetY = np.append(np.ones(1666) , np.zeros(3334))
11
12 ### data to do some KNN on
13 n1dataX = np.ndarray(shape=(30000,3))
14 n1dataX.T[0] = np.append(Signal.x[:10000] , Background.x[:20000])
15 n1dataX.T[1] = np.append(Signal.y[:10000] , Background.y[:20000])
16 n1dataX.T[2] = np.append(Signal.NumberOfHits[:10000] , Background.NumberOfHits[:20000])
17
18 n2dataX = np.ndarray(shape=(30000,3))
19 n2dataX.T[0] = n1dataX.T[0]
20 n2dataX.T[1] = n1dataX.T[1]
21 n2dataX.T[2] = np.append(np.log10(Signal.NumberOfHits[:10000]) , np.log10(Background.
    NumberOfHits[:20000]))

```

plots/class\_structure.py

Signal = Signal[Signal. Acceptance Mask ]  
<sup>2</sup>  
 ↳ Nans rauswerfen

Werte nach folgender Form durch den KNN jagen (einmal für d gezeigt):

```

1 ##### KNN on Data
2 Neutrino1 = KNN(10)
3 Neutrino1.fit(TrainingsetX, TrainingsetY)
4 N1labels = Neutrino1.predict(n1dataX)

```

plots/class\_structure.py

Funktion die Reinheit, Effizienz und Signifikanz bestimmt (Werte dazu in 1):

```

1 def reineffisign(predictions):
2     S = predictions[predictions == 1]
3     B = predictions[predictions == 0]
4     array1 = predictions[:10000] #sollte nur Signal sein
5     array2 = predictions[10000:] #sollte nur Untergrund sein
6     trupos = len(array1[array1 == 1]) #alle Signale zu S
7     truneg = len(array2[array2 == 0]) #Back das zu B
8     falpos = len(array2[array2 == 1]) #Back das zu S
9     falneg = len(array1[array1 == 0]) #Signal zu B
10    rein = (trupos / (truneg + falpos)) f
11    effi = (trupos / (trupos + falneg)) ✓
12    sign = (len(S) / np.sqrt(len(S) + len(B))) f
13    return rein, effi, sign

```

plots/class\_structure.py

In d) sieht man gut was in a) diskutiert wird, da der Untergrund doppelt so groß ist wie das Signal werden nur noch ca. ein Fünftel richtig zu geordnet. Das erklärt warum Reinheit und Effizienz klein ausfallen. Bei der Signifikanz ist iwas schief gelaufen.

Bei der e) sieht man, dass die Anzahl der Hits wesentliches Attribut zur Unterscheidung darstellt. Der Logarithmus sorgt dafür, dass die Werte dahingehend näher zusammenrücken und können nicht mehr unterschieden werden vom Untergrund.

In der f) tritt dann, das so genannte Overfitting ein,  $k = 10$  scheint schon ein guter Wert zu seien, um beide Population zu trennen.  $k = 20$  dagegen scheint dagegen ins Overfitting zu gehen, da Reinheit, Effizienz (und Signifikanz) abnehmen.

Teilaufgabe	Reinheit	Effizienz	Signifikanz
d)	0.0921	0.1842	13.94878250362136
e)	0.0	0.0	0.0
f)	0.08535	0.1707	13.290603196745186

Tabelle 1: Reinheit, Effizienz und Signifikanz für die einzelnen teilaufgaben.

0.5P.

## 2 Aufgabe 2

Teilaufgabe a) und b) sind auf einem separaten Zettel abgegeben worden. Für die Berechnung des Informationsgewinns in Abhängigkeit verschiedener Schnitte wurden zunächst Schnitte gewählt, bei denen immer noch ein Wert der Tabelle über dem großen Schnittwert und ein Wert unter dem geringsten liegt. Die gewählten Schnitte für die drei verbleibenden sind :

$$Cut_{\text{Temperatur}} = [19, 20, 21, 22, 23, 24, 25, 26, 27, 28]$$

$$Cut_{\text{Vorhersage}} = [0.5, 1.5]$$

$$Cut_{\text{Feuchtigkeit}} = [70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]$$

In Abhängigkeit dieser Cuts wurden, wie in der handschriftlichen Abgabe beispielhaft durchgeführt, der Informationsgewinn geberechnet. In Abbildung 1 sind die Informationsgewinne für die einzelnen Cuts dargestellt. Die größten Werte für die drei Attribute sind:

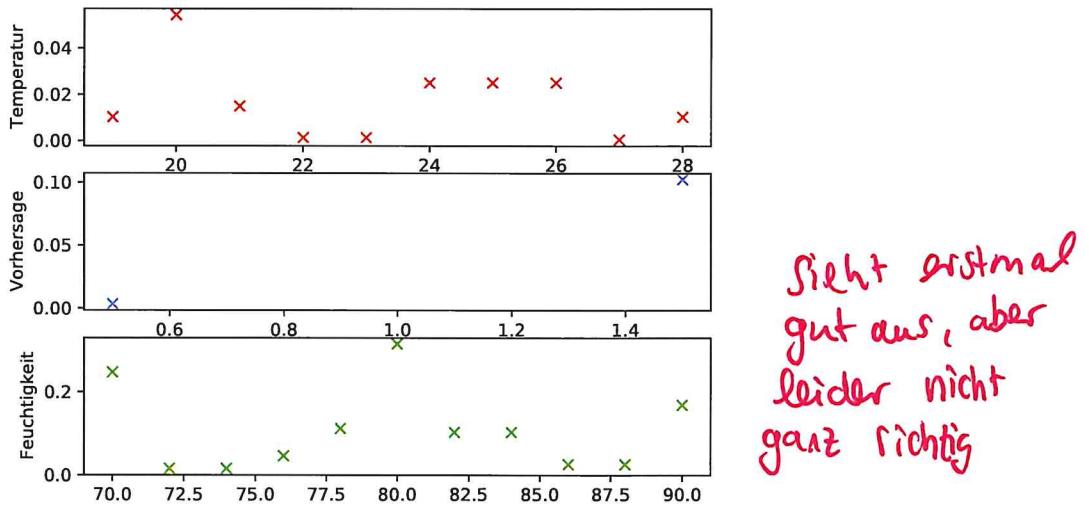


Abbildung 1: Informationsgewinn in Abhängigkeit der Cuts für die einzelnen Attribute.

$$\begin{aligned} IG_{\text{Temperatur}} &= 0.0544 \\ IG_{\text{Vorhersage}} &= 0.1022 \\ IG_{\text{Feuchtigkeit}} &= 0.3149 \end{aligned}$$

*• leider falsch  
• bei welchen cuts erhältet ihr diese Werte*

Demnach lohnt es sich am meisten das Attribut Feuchtigkeit zu verwenden um die Daten zu trennen.

*ff ctd: 2/3,5*

### 3 Aufgabe 4

Die Rechnungen sind auf einem separaten Zettel abgegeben.

*Aufgabe 3 fehlt → 0/3*

# Code fuer Blatt06

Kasper, Appel, Schroeer

30. November 2018

```
../B/1/Blatt06_Kasper_Appel_Schroeer/Blatt6.py
```

```
1 def Aufgabe1():
2     """ one classy boi
3     class KNN:
4         '''KNN Classifier.
5
6             Attributes
7
8                 k : int
9                     Number of neighbors to consider.
10                """
11            def __init__(self, k):
12                """Initialization.
13                    Parameters are stored as member variables/attributes.
14
15                Parameters
16
17                    k : int
18                        Number of neighbors to consider.
19                    """
20                    self.k = k
21                    #Hier fehlt glaube ich noch ein self.traindata und self.trainlabels
22            def fit(self, X, y):
23                """Fit routine.
24                    Training data is stored within object.
25
26                Parameters
27
28                    X : numpy.array, shape=(n_samples, n_attributes)
29                        Training data.
30                    y : numpy.array shape=(n_samples)
31                        Training labels.
32                    """
33                    # Code
34                    self.traindata = X
35                    self.trainlabels = y
36
37            def predict(self, X):
38                """Prediction routine.
39                    Predict class association of each sample of X.
40
41                Parameters
42
43                    X : numpy.array, shape=(n_samples, n_attributes)
44                        Data to classify.
45
46                Returns
47
48                    prediction : numpy.array, shape=(n_samples)
49                        Predictions, containing the predicted label of each sample.
50                    """
51                    prediction = np.array([])
52                    # Code
53                    for x in X:
54                        abst1 = self.traindata.T[0] - x.T[0]
55                        abst2 = self.traindata.T[1] - x.T[1]
56                        abst3 = self.traindata.T[2] - x.T[2]
57                        abst = np.sqrt(abst1**2 + abst2**2 + abst3**2)
```

```

58     indis = np.argsort(abst)
59     indis = indis[:self.k]
60     labels = np.array(self.trainlabels[indis])
61     u, inds = np.unique(labels, return_inverse=True)
62     label = u[np.argmax(np.bincount(inds))]
63     prediction = np.append(prediction,label)
64     return prediction
65     #####read in some data
66 Background = pd.read_hdf('NeutrinoMC.hdf5', key= 'Background')
67 Signal = pd.read_hdf('NeutrinoMC.hdf5', key= 'Signal')
68     ### Some training data
69 TrainingsetX = np.ndarray(shape=(5000,3))
70 TrainingsetX.T[0] = np.append(Signal.x[:1666],Background.x[:3334])
71 TrainingsetX.T[1] = np.append(Signal.y[:1666],Background.y[:3334])
72 TrainingsetX.T[2] = np.append(Signal.NumberOfHits[:1666],Background.NumberOfHits[:3334])
73 TrainingsetY = np.append(np.ones(1666),np.zeros(3334))
74     ### data to do some KNN on
75 n1dataX = np.ndarray(shape=(30000,3))
76 n1dataX.T[0] = np.append(Signal.x[:10000],Background.x[:20000])
77 n1dataX.T[1] = np.append(Signal.y[:10000],Background.y[:20000])
78 n1dataX.T[2] = np.append(Signal.NumberOfHits[:10000],Background.NumberOfHits[:20000])
79
80 n2dataX = np.ndarray(shape=(30000,3))
81 n2dataX.T[0] = n1dataX.T[0]
82 n2dataX.T[1] = n1dataX.T[1]
83 n2dataX.T[2] =
84     np.append(np.log10(Signal.NumberOfHits[:10000]),np.log10(Background.NumberOfHits[:20000]))
85
86     ##### KNN on Data
87 Neutrino1 = KNN(10)
88 Neutrino1.fit(TrainingsetX,TrainingsetY)
89 N1labels = Neutrino1.predict(n1dataX)
90
91 Neutrino2 = KNN(10)
92 Neutrino2.fit(TrainingsetX,TrainingsetY)
93 N2labels = Neutrino2.predict(n2dataX)
94
95 Neutrino3 = KNN(20)
96 Neutrino3.fit(TrainingsetX,TrainingsetY)
97 N3labels = Neutrino3.predict(n1dataX)
98
99     #### reinheit Effizienz und Signifikanz
100 def reineffisign(predictions):
101     S = predictions[predictions == 1]
102     B = predictions[predictions == 0]
103     array1 = predictions[:10000] #sollte nur Signal sein
104     array2 = predictions[10000:] #sollte nur Untergrund sein
105     trupos = len(array1[array1 == 1]) #alle Signale zu S
106     truneg = len(array2[array2 == 0]) # Back das zu B
107     falpos = len(array2[array2 == 1]) #Back das zu S
108     falneg = len(array1[array1 == 0]) # Signal zu B
109     rein = (trupos/(truneg + falpos)) ↑
110     effi = (trupos/(trupos + falneg)) ↑
111     sign = (len(S) / np.sqrt(len(S) + len(B))) {
112     return rein,effi,sign
113
114 print(reineffisign(N1labels))
115 print(reineffisign(N2labels))
116 print(reineffisign(N3labels))
117
118 def Aufgabe2():
119     # Diese Tabelle habe ich abgetippt, hoffentlich ohne Fehler :D
120     Temperatur = np.array([29.4, 26.7, 28.3, 21.1, 20.0, 18.3, 17.8, 22.2, 20.6,
121                           23.9, 23.9, 22.2, 27.2, 21.7])
122     Vorhersage = np.array([2, 2, 1, 0, 0, 0, 1, 2, 2, 0, 2, 1, 1, 0])
123     Feuchtigkeit = np.array([85, 90, 78, 96, 80, 70, 65, 95, 70, 80, 70,
124                            90, 75, 80])
125     Wind = np.array([0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1])
126     Fussball = np.array([0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0])
127
128     # Definitionen und bums

```

```

128 def Entropie(zp, zn, z):
129     wertP = zp / z
130     wertN = zn / z
131     if (wertP != 0 and wertN != 0):
132         return(-wertP*np.log2(wertP)-wertN*np.log2(wertN))
133     if (wertP == 0 and wertN != 0):
134         return(-wertN*np.log2(wertN))
135     if (wertP != 0 and wertN == 0):
136         return(-wertP*np.log2(wertP))
137
138 def IG(testarray, zielarray, cut, Hz):
139     nTruePos = 0
140     nTrueNeg = 0
141     nFalsePos = 0
142     nFalseNeg = 0
143     lenPos = len(testarray[testarray > cut])
144     lenNeg = len(testarray[testarray < cut])
145     lenGes = len(zielarray)
146     for i in np.arange(len(testarray)):
147         if (testarray[i] > cut and zielarray[i] == 1):
148             nTruePos += 1
149         if (testarray[i] > cut and zielarray[i] == 0):
150             nTrueNeg += 1
151         if (testarray[i] < cut and zielarray[i] == 1):
152             nFalseNeg += 1
153         if (testarray[i] < cut and zielarray[i] == 0):
154             nFalsePos += 1
155     Htrue = Entropie(nTruePos, nTrueNeg, lenPos)
156     Hfalse = Entropie(nFalsePos, nFalseNeg, lenNeg)
157     Gain = Hz - ((lenPos/lenGes)*Htrue + (lenNeg/lenGes)*Hfalse)
158     return(Gain)
159
160 # Entropie der Wurzel des Baumes
161 HBaum = Entropie(len(Fussball[Fussball == 1]), len(Fussball[Fussball == 0]),
162                   len(Fussball))
163
164 # gewählte Cuts, so dass immer Werte drüber und drunter liegen
165 Temperaturcuts = [19, 20, 21, 22, 23, 24, 25, 26, 27, 28]
166 Vorhersagecuts = [0.5, 1.5]
167 Feuchtigkeitcus = [70, 72, 74, 76, 78, 80, 82, 84, 86, 88, 90]
168
169 # Informationsgewinn für die cuts testen
170 GainT = []
171 GainV = []
172 GainF = []
173 for c in Temperaturcuts:
174     GainT.append(IG(Temperatur, Fussball, c, HBaum))
175 for d in Vorhersagecuts:
176     GainV.append(IG(Vorhersage, Fussball, d, HBaum))
177 for e in Feuchtigkeitcus:
178     GainF.append(IG(Feuchtigkeit, Fussball, e, HBaum))
179
180 # größten Informationsgewinn finden
181
182 bestT = np.argsort(GainT)[-1]
183 bestV = np.argsort(GainV)[-1]
184 bestF = np.argsort(GainF)[-1]
185 print(GainT[bestT])
186 print(GainV[bestV])
187 print(GainF[bestF])
188
189 # Plotten des Informationsgewinns
190
191 fig, (ax1, ax2, ax3) = plt.subplots(3, 1)
192 ax1.plot(Temperaturcuts, GainT, 'rx')
193 ax1.set_ylabel('Temperatur')
194 ax2.plot(Vorhersagecuts, GainV, 'bx')
195 ax2.set_ylabel('Vorhersage')
196 ax3.plot(Feuchtigkeitcus, GainF, 'gx')
197 ax3.set_ylabel('Feuchtigkeit')
198 plt.savefig('Informationsgewinn.pdf')

```

```

129
130
131 if __name__ == '__main__':
132     import matplotlib.pyplot as plt
133     import numpy as np
134     import pandas as pd
135
136     Aufgabe1()
137     Aufgabe2()

```

## A 77 a) Umgang mit nicht Num. Datentyp.

in ~~wicht~~ Num. Datentypen umwandeln

z.B. "Hallo" = ? usw.

mit Aussagekräftigen Attributen arbeiten

z.B. nicht sondern Rethorik o.ä.  
Groß und  
Kleinschr.

Attribut (nicht sortierbar)	→			neu zuordnen
	A	B	C	
A	1	0	0	
B	0	1	0	
C	0	0	1	
A	1	0	0	

## b) Normieren:

hilft beim Vergleichen von Dingen

## c) Lücken: (z.B. NaN, Inf)

- Fehlerhaften Wert durch MW der Spalte oder der Nachbam
- den Häufigsten Wert (für kategorisch = Strings etc...)
- den "Fehler" wählen, der am wenigsten schadet

## d) Zusammenfügen:

Attribute müssen zusammen passen, Größenordnungen, ...  
redundante Daten aussortieren

- e) Attribute entf. überflüssige z.B. ~~zu~~ wenig Informationsgehalt. PCA hilft  
hohe Korrelationen für Attribute = ähnlich. Informationen eins wegschm.

# SMD Blatt 6

## Aufgabe 16

Julian Schröer  
Robert Appel  
Yvonne Kasper

a) Entropie der Wurzel bestimmen:

$$H(Y) = - \sum_{z \in Z} P(Y=z) \log_2(P(Y=z))$$

$Y$  entspricht dem Zielattribut Fußball

$z \in Z$  wobei  $Z = [\text{true}, \text{false}]$

Es gibt 14 Werte, davon 9 mal „true“ und 5 mal „false“.

$$\begin{aligned} \Rightarrow H(Y) &= - \frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) \\ &= - \frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) \\ &\approx 0,9403 \quad \checkmark \quad 0,5/0,5 \end{aligned}$$

b) Informationsgewinn für den Schnitt auf dem Attribut Wind berechnen:

$$IG(X, Y) = H(Y) - H(Y|X)$$

$$\text{mit } H(Y|X) = - \sum_{m \in M} P(X=m) \sum_{z \in Z} P(Y=z|X=m) \log_2(P(Y=z|X=m))$$

$X$  entspricht dem Attribut Wind mit  $M = [\text{true}, \text{false}]$ .

Wind ist 6 mal true  $\begin{cases} \text{mit 3 mal Fußball false} \\ \text{mit 3 mal Fußball true} \end{cases}$

Wind ist 8 mal false  $\begin{cases} \text{mit 2 mal Fußball false} \\ \text{mit 6 mal Fußball true} \end{cases}$

$$\begin{aligned} H(Y|X=\text{true}) &= - \frac{3}{6} \log_2\left(\frac{3}{6}\right) - \frac{3}{6} \log_2\left(\frac{3}{6}\right) \\ &= 1 \end{aligned}$$

$$\begin{aligned} H(Y|X=\text{false}) &= - \frac{2}{8} \log_2\left(\frac{2}{8}\right) - \frac{6}{8} \log_2\left(\frac{6}{8}\right) \\ &\approx 0,8113 \quad \checkmark \end{aligned}$$

$$\Rightarrow IG(X, Y) = 0,9403 - \left( \frac{6}{14} \cdot 1 + \frac{8}{14} \cdot 0,8113 \right) \approx 0,0481 // \quad \checkmark \quad 2/2$$

A17) e) Spalten mit nur Nullen...  
=> keine Infos enthalten  
Laufzeitoptimierung

e) Simulationsspaß:  
keine Attribute in Simulation  
tun die man in Daten nicht hat.  
z.B. Label

# Aufgabe 18

a) Definition der bedingten Wahrscheinlichkeit:

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (*)$$

$$P(F|W) = \frac{P(F \cap W)}{P(W)} = \frac{P(W \cap F)}{P(W)} = \frac{P(W|F) \cdot P(F)}{P(W)} //$$

↑ Einsetzen  
 der Definition (\*)      ↑  $P(W \cap F)$   
 ↓                              ↓  $P(F \cap W)$       ↑ Einsetzen  
 der Definition (\*)      ↓

b) Wie hoch ist die Wahrscheinlichkeit heute Fußball zu spielen?

Wind	F=ja	F=nein
schwach	6	2
stark	3	3

Feuchtigkeit	F=ja	F=nein
normal	6	1
hoch	3	4

Temperatur	F=ja	F=nein
kalt	3	3
mild	6	1
heiß	0	1

Ausblick	F=ja	F=nein
regnerisch	3	3
bewölkt	4	1
sonnig	2	1

Heute: Wind stark  
 Feuchtigkeit hoch  
 Temperatur kalt  
 Ausblick sonnig

Insgesamt  
 ↗ 9 mal ja  
 ↘ 5 mal nein

$$P(W|F=ja) = \prod_i P(x_i | F=ja)$$

(Tipp 1)

$$= P(x=\text{stark} | F=ja) \cdot P(x=\text{hoch} | F=ja) \cdot P(x=\text{kalt} | F=ja)$$

$$\cdot P(x=\text{sonnig} | F=ja)$$

$$= \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{2}{9}$$

$$= \frac{54}{6561} = \frac{2}{243} \approx 0.00823 \approx 0,8\%$$

zur Normierung:

Wie ist die Wahrscheinlichkeit für Fußball: ja / nein bei den heutigen Bedingungen.

$$P(\text{ja}) P(W| \text{ja}) = P(F=\text{ja}) \cdot P(x=\text{stark} | \text{ja}) \cdot P(x=\text{hoch} | \text{ja}) \cdot P(x=\text{kalt} | \text{ja}) \cdot P(x=\text{sonnig} | \text{ja}) \\ = \frac{9}{14} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{3}{9} \cdot \frac{2}{9} = \frac{1}{189} = 0,00529$$

$$P(\text{nein}) P(W| \text{nein}) = P(F=\text{nein}) \cdot P(x=\text{stark} | \text{nein}) \cdot P(x=\text{hoch} | \text{nein}) \cdot P(x=\text{kalt} | \text{nein}) \cdot P(x=\text{sonnig} | \text{nein}) \\ = \frac{5}{14} \cdot \frac{3}{5} \cdot \frac{4}{5} \cdot \frac{3}{5} \cdot \frac{1}{5} = \frac{18}{875} = 0,020$$

$$\Rightarrow P(W) = P(F=\text{ja}) \cdot P(W|F=\text{ja}) + P(F=\text{nein}) P(W|F=\text{nein}) \quad \checkmark \text{ sehr gut!} \\ = \frac{1}{189} + \frac{18}{875} = 0,0259$$

Um also insgesamt die Wahrscheinlichkeit zu berechnen, hätte Fußball zu spielen:

$$P(F=\text{ja} | W) = \frac{P(W|F=\text{ja}) \cdot P(F=\text{ja})}{P(W)} \\ = \frac{\frac{1}{189}}{\frac{611}{23625}} = \frac{0,00529}{0,0259} = 0,2042 \\ \approx 20\% \quad \checkmark \quad 2/2$$

c) Um für den morgen Tag die Wahrscheinlichkeit zu berechnen betrachten wird die Daten:

Wind : schwach  
Feuchtigkeit : hoch  
Temperatur : heiß  
Ausblick : sonnig

Problem: In den (doch eher wenigen Daten) ist kein Fall von Temperatur = heiß und Fußball = ja vorliegt. Dies gebe eine Wahrscheinlichkeit von 0.  $\checkmark$

Lösungsidee: Nur die anderen Attribute betrachten?!  
+ Laplace Korrektur

$1/1 \Rightarrow 1$