blatt06_nitschke_grisard

November 29, 2018

In [1]:
```
import numpy as np
import pandas as pd
from pandas import DataFrame, Series
from collections import Counter
import matplotlib.pyplot as plt

from ml import plots
%matplotlib inline
from ml import plots
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
from IPython.display import SVG
from graphviz import Source
from IPython.display import display
from ipywidgets import interactive
from matplotlib.colors import ListedColormap
```

Die Klassenstruktur:

In [2]:
```
class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.training_data = X
        self.training_labels = y

    def predict(self, X):
        #calculate the eucleadian distance (ignore the root, cause its a monoton funct
        #between each test event and each training event
        distance = (-2 * np.dot(X, self.training_data.T)
                    + np.sum(X**2, axis=1)[:, np.newaxis]
                    + np.sum(self.training_data**2, axis=1)[np.newaxis, :])

        #generate matrix with labels of the k nearest neighbours
```

```
            labels = self.training_labels.values[(np.argsort(distance))[:, :self.k]]

            #most common label of the k nearest neighbours as prediction for each test eve
            prediction = []
            for i in range(np.shape(labels)[0]):
                count = Counter(labels[i, :])
                prediction.append(count.most_common(1)[0][0])

            return prediction
```

Bringe zunächst die Daten in die benötigte Form:

```
In [3]: #read hdf5 file
        neutrino_signal = pd.read_hdf('NeutrinoMC.hdf5', key = 'Signal')

        #select the accepted events
        neutrino_signal = neutrino_signal[neutrino_signal.AcceptanceMask]

        #delete the energy and the acceptancemask (not relevant for this task)
        neutrino_signal = neutrino_signal.drop(columns = ['Energy', 'AcceptanceMask'])

        #reset the index of the DataFrame
        neutrino_signal = neutrino_signal.reset_index(drop = True)

        #add label to the signal events
        neutrino_signal['label'] = Series(data = ['signal' for i in neutrino_signal.x])
```

Das gleiche für die Untergrundevents:

```
In [4]: neutrino_background = pd.read_hdf('NeutrinoMC.hdf5', key = 'Background')
        neutrino_background['label'] = Series(data = ['background' for i in neutrino_backgroun
```

Eine Funktion um einen gewünschten gemischten Datensatz aus Signal und Untergrund zu erstellen:

```
In [5]: def mix_sample(signal_events, background_events, n_signal, n_background):
            data_set = pd.concat([background_events.sample(n_background), signal_events.sample
            X = data_set.drop(columns = 'label')
            y = data_set['label']
            return X, y
```

Funktionen für Reinheit usw:

```
In [6]: #Reinheit
        def precision(true_pos, false_pos):
            return len(true_pos) / (len(true_pos) + len(false_pos))

        #Effizienz
        def recall(true_pos, false_neg):
```

```
        return len(true_pos) / (len(true_pos) + len(false_neg))

    #Signifikanz
    def significance(signal, noise):
        return len(signal) / np.sqrt(len(noise) + len(signal))
```

Generiere den Trainings- und Testdatensatz:

```
In [7]: X_training, y_training = mix_sample(neutrino_signal, neutrino_background, 5000, 5000)
        X_test, y_test = mix_sample(neutrino_signal, neutrino_background, 10000, 20000)
```

Ab hier ist das Vorgehen für die Aufgabenteile d)-f) analog, wesegen eine Funktion für die Prozedur geschrieben wird:

```
In [8]: def procedure(k, X_training, y_training, X_test, y_test):
            #use the knn algorithm
            knn = KNN(k = k)
            knn.fit(X = X_training, y = y_training)
            prediction = knn.predict(X = X_test)

            #add results to test data set
            X_test['prediction'] = Series(prediction)
            X_test['truth'] = y_test


            #calculate true positive etc
            true_positive  = X_test[(X_test.truth == 'signal') & (X_test.prediction == 'signal
            true_negative  = X_test[(X_test.truth == 'background') & (X_test.prediction == 'bac
            false_positive = X_test[(X_test.truth == 'background') & (X_test.prediction == 'sig
            false_negative = X_test[(X_test.truth == 'signal') & (X_test.prediction == 'backgro

            #calculate precision etc
            precision_knn = precision(true_positive, false_positive)
            recall_knn = recall(true_positive, false_negative)
            significance_knn = significance(X_test[X_test.prediction == 'signal'], X_test[X_tes

            print(f'Reinheit: \t{precision_knn}\nEffizienz: \t{recall_knn} \nSignifikanz: \t{s
```

Hier nun KNN Algorithmus mit $k = 10$:

```
In [9]: procedure(11, X_training, y_training, X_test, y_test)
```

```
Reinheit:          0.829541566883339
Effizienz:          0.9699
Signifikanz:          67.50379347365104
```

e) Nun $log_{10}(N)$ statt N:

```
In [10]: neutrino_signal_e = neutrino_signal
         neutrino_signal_e['log10NumberOfHits'] = np.log10(neutrino_signal['NumberOfHits'])
         neutrino_signal_e = neutrino_signal_e.drop(columns = 'NumberOfHits')

         neutrino_background_e = neutrino_background
         neutrino_background_e['log10NumberOfHits'] = np.log10(neutrino_background['NumberOfHi
         neutrino_background_e = neutrino_background.drop(columns = 'NumberOfHits')
```

Nun das gleiche wie oben:

```
In [11]: X_training, y_training = mix_sample(neutrino_signal_e, neutrino_background_e, 5000, 50
         X_test, y_test = mix_sample(neutrino_signal_e, neutrino_background_e, 10000, 20000)

In [12]: procedure(10, X_training, y_training, X_test, y_test)

Reinheit:         0.8737274513305947
Effizienz:         0.9784
Signifikanz:         64.6516831438543
```

Kommentar: Die Reinheit ist etwas höher.
f) Nun das ganze mit $k = 20$:

```
In [13]: X_training, y_training = mix_sample(neutrino_signal, neutrino_background, 5000, 5000)
         X_test, y_test = mix_sample(neutrino_signal, neutrino_background, 10000, 20000)

In [14]: procedure(20, X_training, y_training, X_test, y_test)

Reinheit:         0.8249143835616438
Effizienz:         0.9635
Signifikanz:         67.43451144134829
```

0.1 Aufgabe 16

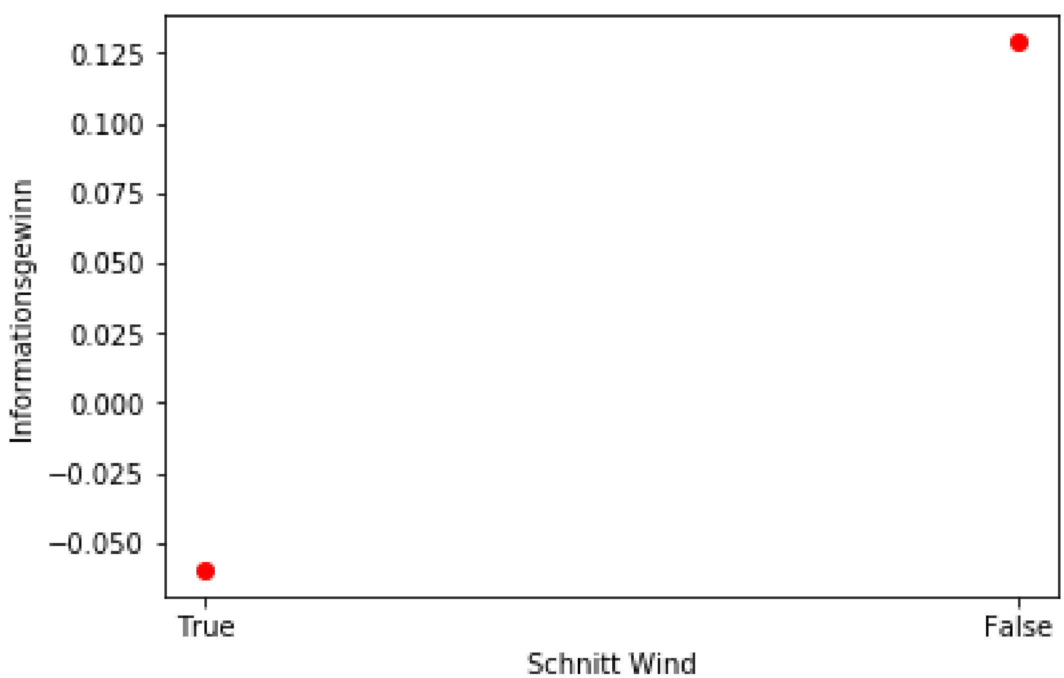
```
In [15]: X = DataFrame()
         X['temperature'] = Series([29.4, 26.7, 28.3, 21.1, 20, 18.3, 17.8, 22.2, 20.6, 23.9, 2
         X['report'] = Series([2, 2, 1, 0, 0, 0, 1, 2, 2, 0, 2, 1, 1, 0])
         X['humidity'] = Series([85, 90, 78, 96, 80, 70, 65, 95, 70, 80, 70, 90, 75, 80])
         X['wind'] = Series([False, True, False, False, False, True, True, False, False, False
         #X

         y = DataFrame({'football': Series([False, False, True, True, True, False, True, False

In [16]: def entropy(y, splits = [[True for y in range(len(y))]]):
             H = []
             for split in splits:
                 entropy = 0
                 values = Counter(y[split]).most_common()
```

```
        for value in values:
            entropy -= value[1] / len(y[split]) * np.log2(value[1] / len(y[split]))
        H.append(entropy)
    return np.array(H)

def information_gain(y, splits):
    return entropy(y) - entropy(y, splits)
```

In [17]:
```
print(entropy(y.football))
```

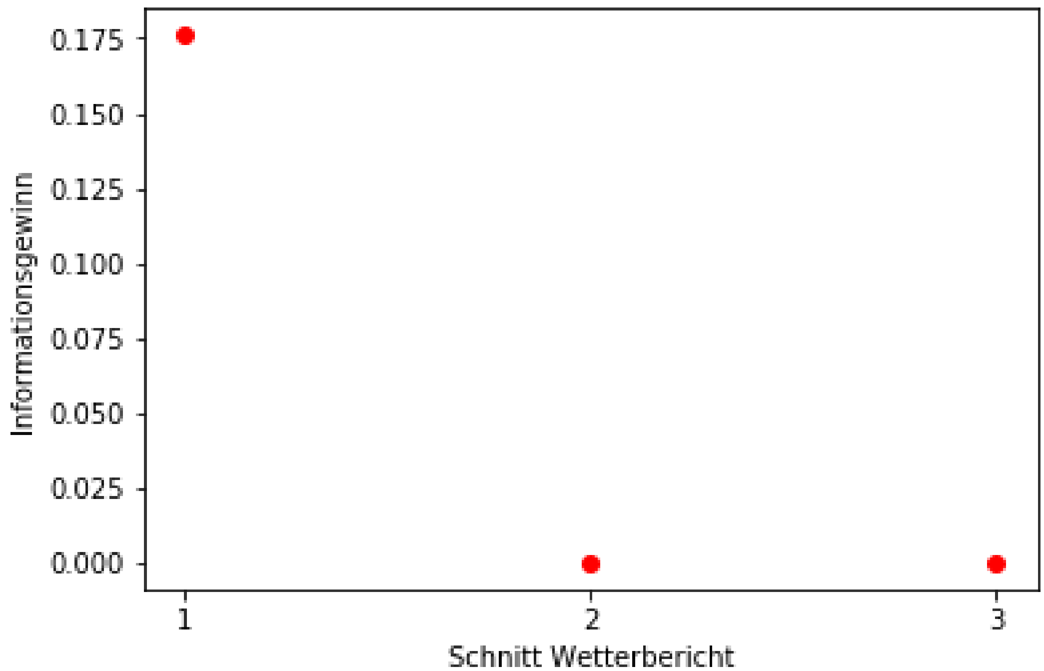
```
[0.94028596]
```

In [18]:
```
wind_splits = [X.wind == True, X.wind == False]

wind_information_gain = information_gain(y.football, wind_splits)
plt.plot([1, 2], wind_information_gain, 'ro')
plt.xticks([1, 2], ['True', 'False'])
plt.xlabel('Schnitt Wind')
plt.ylabel('Informationsgewinn')
None
```

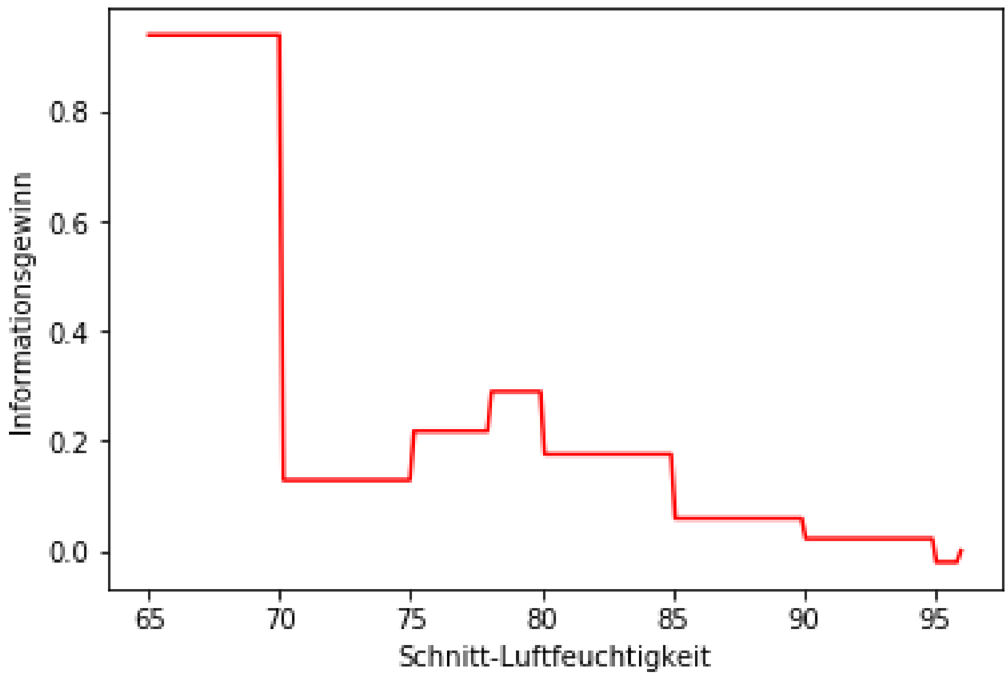


In [19]:
```
report_split = [1, 2, 3]
report_splits = [X.report <= report for report in report_split]
```

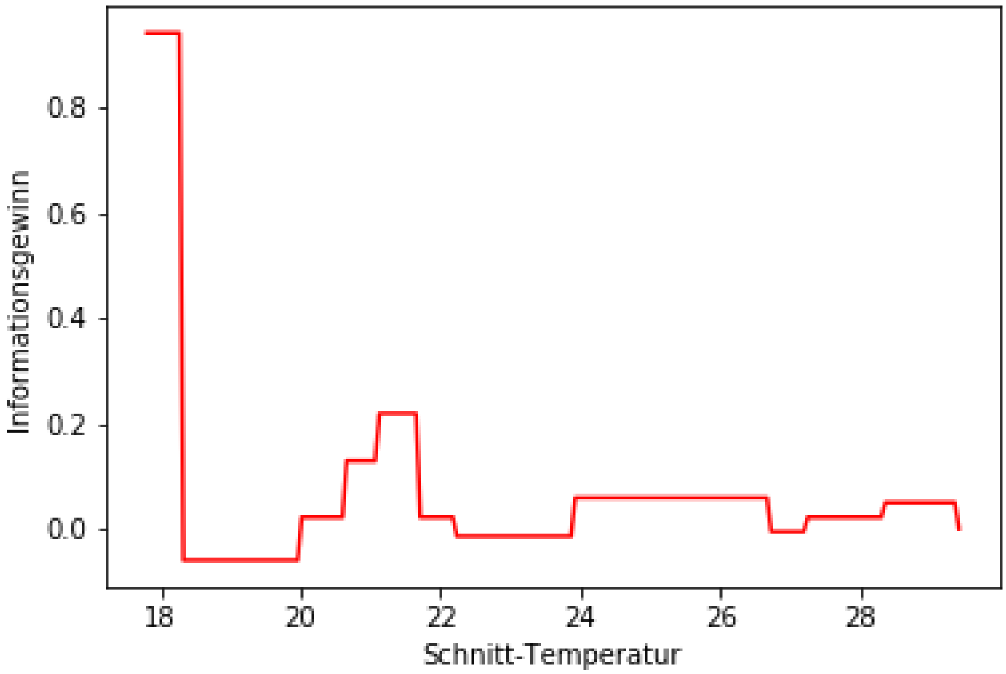
```
report_information_gain = information_gain(y.football, report_splits)
plt.plot(report_split, report_information_gain, 'ro')
plt.xticks([1, 2, 3])
plt.xlabel('Schnitt Wetterbericht')
plt.ylabel('Informationsgewinn')
None
```



In [20]:
```
humidity_split = np.linspace(min(X.humidity), max(X.humidity), 200)
humidity_splits = [X.humidity <= H for H in humidity_split]

H_information_gain = information_gain(y.football, humidity_splits)
plt.plot(humidity_split, H_information_gain, 'r-')
plt.xlabel('Schnitt-Luftfeuchtigkeit')
plt.ylabel('Informationsgewinn')
None
```




```
In [21]: temperature_split = np.linspace(min(X.temperature), max(X.temperature), 200)
         temperature_splits = [X.temperature <= T for T in temperature_split]


         T_information_gain = information_gain(y.football, temperature_splits)
         plt.plot(temperature_split, T_information_gain, 'r-')
         plt.xlabel('Schnitt-Temperatur')
         plt.ylabel('Informationsgewinn')
         None
```



In [22]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

discrete_cmap = ListedColormap(['xkcd:red', 'xkcd:blue'])
clf = DecisionTreeClassifier(max_depth=10, criterion='entropy')
clf.fit(X, y)
tree.export_graphviz(clf, out_file = 'tree.dot')
```

SMD Blatt 6

A16

a) $H(S)$: Fußball ins. 14 Tage → 9 gespielt
→ 5 nicht

$$H(S) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) \approx 0.940286$$

b) Wind = schwach: 8x → 6 gespielt
→ 2 nicht

$$H(W_{schw}) = -\frac{6}{8}\log_2\left(\frac{6}{8}\right) - \frac{2}{8}\log_2\left(\frac{2}{8}\right) = 0.8112781$$

Wind = stark: 6x → 3 gespielt
→ 3 nicht

$$H(W_{stark}) = -\frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1$$

$$Gain(W_{schw}) = 0.940286 - 0.811278 = 0{,}129008$$

$$Gain(W_{stark} = 0{,}940286 - 1 = -0{,}059714$$

Gesamt: $Gain(Wind) = H(S) - \frac{8}{14}H(W_{schw}) - \frac{6}{14}H(W_{stark})$

$= 0.048127$

c) / d): siehe ipynb

A18

a) $$P(A|B) = \frac{P(A\cap B)}{P(B)} = \frac{P(B\cap A)}{P(B)} = \frac{P(B|A)\,P(A)}{P(B)}$$

b) $P(W|F) = P(W_1|F)\,P(W_2|F)\,P(W_3|F)\,P(W_4|F)$

$P(W) = P(W_1)P(W_2)P(W_3)P(W_4)$ da W_i unabhängig

gesucht P(ja| stark, hoch, kalt, sonnig)

$P(\text{stark}|\text{ja}) = 1/3$ $P(\text{hoch}|\text{ja}) = 1/3$ $P(\text{kalt}|\text{ja}) = 1/3$

$P(\text{sonnig}|\text{ja}) = 2/9$ $P(\text{ja}) = 9/14$

$P(\text{stark}) = 3/7$ $P(\text{hoch}) = 1/2$ $P(\text{kalt}) = 3/7$

$P(\text{sonnig}) = 3/14$

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

$$P(W|F) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{2}{9} = \frac{2}{243} \qquad P(F) = 9/14$$

$$P(W) = 3/7 \cdot 1/2 \cdot 3/7 \cdot \frac{3}{14} = \frac{27}{1372}$$

$$\Rightarrow P(F|W) = \frac{196}{729} \approx 0{,}27$$

$\Rightarrow$ Die Wahrscheinlichkeit, dass Fußball gespielt wird beträgt 27 %

c) $P(\text{heiß}|\text{ja}) = 0$

$\rightarrow$ Wahrscheinlichkeit wäre direkt null

$\rightarrow$ Berechne stattdessen ~~P(heiß|nein)~~ $P(\text{nein}|W_{\text{morgen}})$

$$P(W_{\text{morgen}}|\text{nein}) = 2/5 \cdot 4/5 \cdot 1/5 \cdot 1/5 = \frac{8}{625}$$

$$P(W_{\text{morgen}}) = 4/7 \cdot 1/2 \cdot 1/14 \cdot 3/14 = \frac{3}{686}$$

$$P(\text{nein}) = 5/14$$

$$P(\text{nein}|W_{\text{morgen}}) = \frac{\frac{8}{625} \cdot \frac{5}{14}}{\frac{3}{686}} \approx 0{,}976$$

$$\Rightarrow P(\text{ja}|W_{\text{morgen}}) = 1 - P(\text{nein}|W_{\text{morgen}}) \approx 2{,}4\ \%$$

andere Möglichkeit:

$$P(\text{ja}|\text{schwach} \wedge \text{hoch} \wedge \text{sonnig})$$

$$= \frac{2/3 \cdot 1/3 \cdot 2/9 \cdot 9/14}{4/7 \cdot 1/2 \cdot 3/14} = \frac{14}{27} \Rightarrow \approx 52\ \%$$