

$$\begin{array}{c|c|c|c|c}
 15 & 16 & 17 & 18 & \\
 \hline
 7/7 & 5/6 & 2,5/3 & 3/4 & 17,5/20
 \end{array}$$

blatt06_nitschke_grisard

November 29, 2018

1 Aufgabe 15

a) Da beim kNN -Algorithmus muss die (euklidische) Norm zwischen den Test und Trainingsdatenpunkten berechnet werden. Weichen die Größenordnungen der Attribute stark voneinander ab, kann es z.B. zu Rundungsfehlern kommen. Zudem ist der Abstandsbegriff sinnvoller, wenn die einzelnen Skalen normiert werden, da dann relative Abstände für alle Attribute gleich gewichtet werden. ✓

b) Der Algorithmus wird als *lazy learner* bezeichnet, da er eigentlich gar nicht lernt. Der Trainingsdatensatz wird einfach nur abgespeichert und für jeden neuen Testdatensatz die k nächsten Nachbarn neu berechnet werden. Die Laufzeit der Lernphase ist verschwindend klein, während die Anwendungsphase lang dauert. Bei der *SVM* ist es genau andersherum.

c)

0,5 P.

```
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 graphviz import Source
        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 euclidean distance (ignore the root,
            #cause its a monoton function)
            #between each test event and each training event
```

sieht gut aus
2.5P.

```

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 event
prediction = []
for i in range(np.shape(labels)[0]):
    count = Counter(labels[i, :])
    prediction.append(count.most_common(1)[0][0])

return prediction

```

d) 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_background.x])
```

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(n_signal)],
                         ignore_index=True)
    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(true_pos, false_pos):
    return len(true_pos) / np.sqrt(len(true_pos) + len(false_pos))
```

Generiere den Trainings- und Testdatensatz:

```
In [7]: X_training, y_training = mix_sample(neutrino_signal, neutrino_background, 5000, 500
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 == 'background')]

    false_positive = X_test[(X_test.truth == 'background') &
                           (X_test.prediction == 'signal')]

    false_negative = X_test[(X_test.truth == 'signal') &
                           (X_test.prediction == 'background')]

    #calculate precision etc
    precision_knn = precision(true_positive, false_positive)
    recall_knn = recall(true_positive, false_negative)
```

```

significance_knn = significance(true_positive, false_positive)

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

```

Hier nun KNN Algorithmus mit $k = 10$:

```
In [9]: procedure(10, X_training, y_training, X_test, y_test)
```

```
Reinheit: 0.8294626788927929
Effizienz: 0.9679
Signifikanz: 89.60116778816749
```

✓ 1 P.

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['NumberOfHits'])
neutrino_background_e = neutrino_background_e.drop(columns = 'NumberOfHits')
```

Nun das gleiche wie oben:

```
In [11]: X_training, y_training = mix_sample(neutrino_signal_e,
                                             neutrino_background_e,
                                             5000, 5000)
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: 0.8692259952123416
Effizienz: 0.9804
Signifikanz: 92.31409240772395
```

✓ 0,5 P.

Kommentar: Alle Attribute werden besser. Wie oben erwähnt ist dies durch eine Um-skalierung des Attributs 'Hits' bedingt. ✓

f) Nun das ganze mit $k = 20$:

```
In [13]: X_training, y_training = mix_sample(neutrino_signal, neutrino_background, 5000, 50
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: 0.8204801900398745
Effizienz: 0.9671
Signifikanz: 89.07785312789944
```

✓ 0,5 P

Kommentar: Alle Attribute werden schlechter. ✓

2 Aufgabe 16

Der handschriftliche Teil der Aufgabe befindet sich am Ende der pdf.

```
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.
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, Fa
#X

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

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)

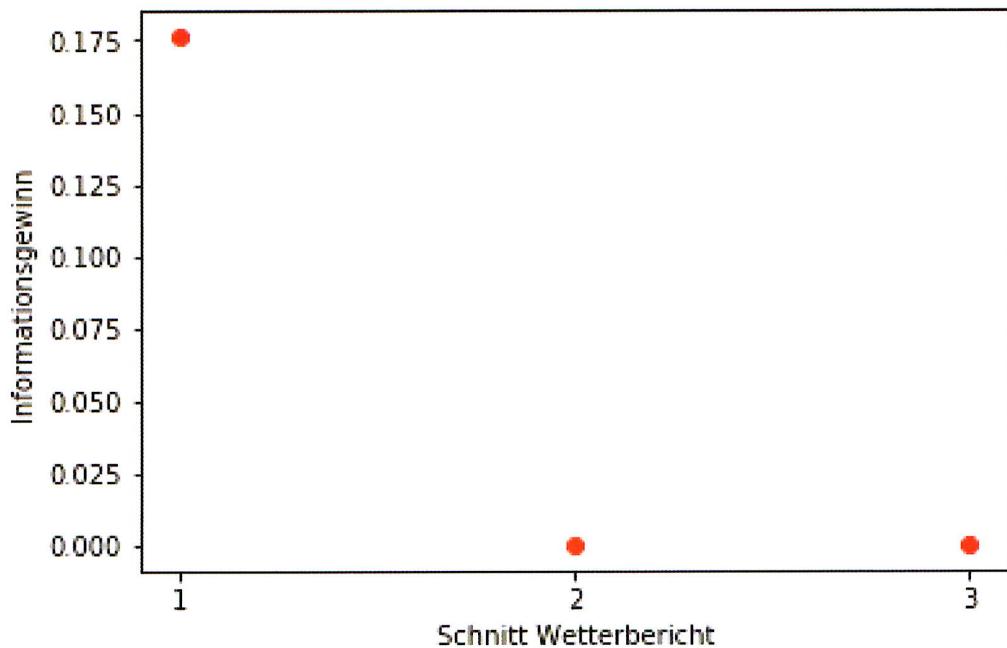
In [17]: print(entropy(y.football))

[0.94028596] ✓
```

```
In [18]: report_split = [1, 2, 3]
report_splits = [X.report <= report for report in report_split]

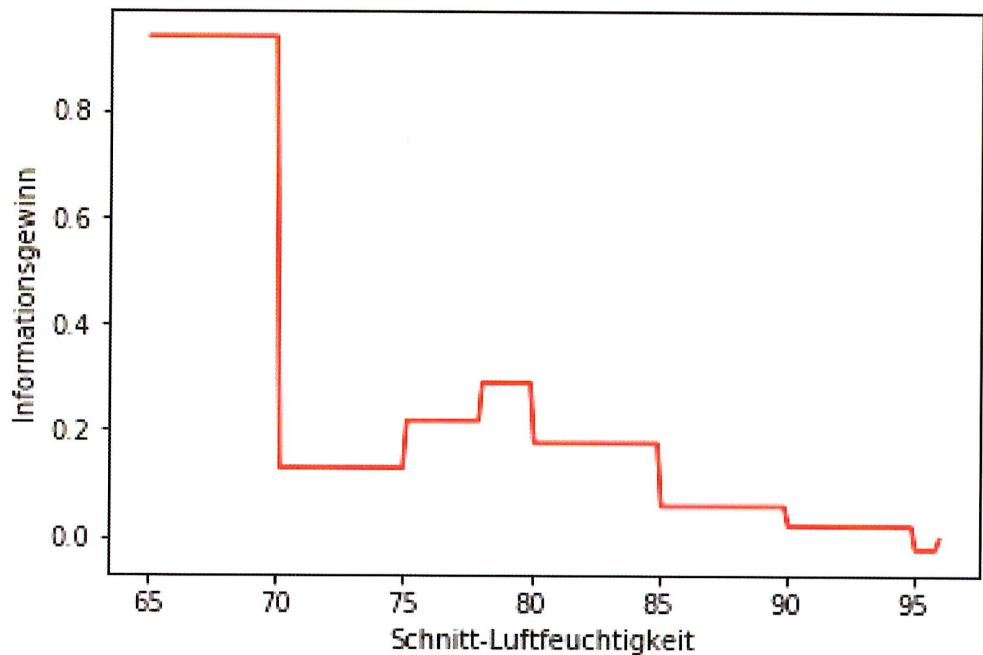
report_information_gain = information_gain(y.football, report_splits)
plt.plot(report_split, report_information_gain, 'ro')
plt.xticks([1, 2, 3])
plt.xlabel('Schnitt Wetterbericht')
plt.ylabel('Informationsgewinn')
None
```

habe versucht euren Code nachzuholfen
haben den auch abgetippt also steigt da nicht so ganz durch
bei mir kommen auf jeden Fall andere Werte für
die Information Gain raus
es scheint also irgendwo bei euch was schief zu laufen
da eure Werte viel höher sind



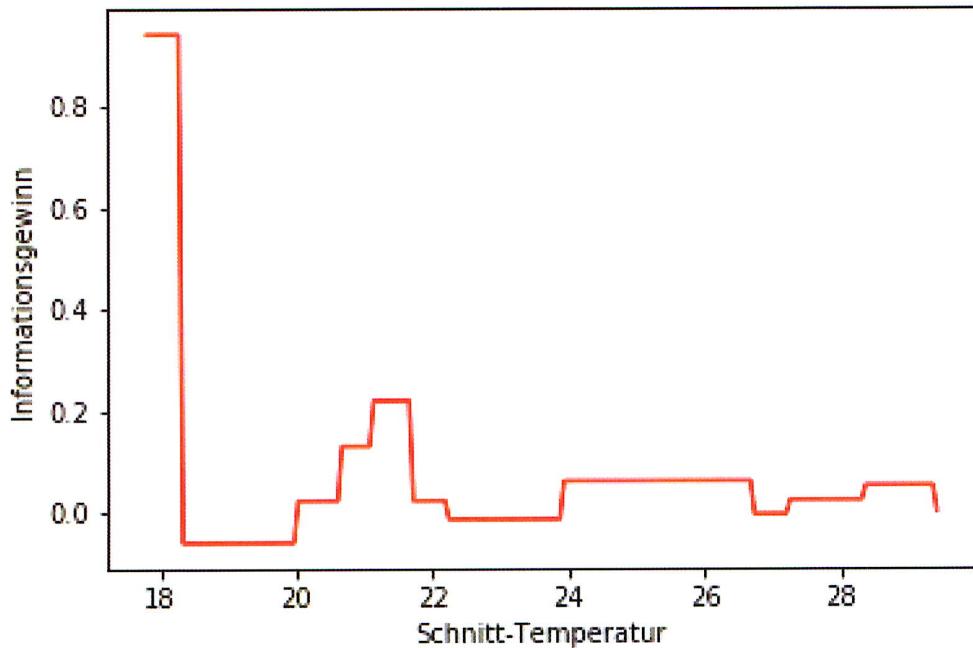
```
In [19]: humidity_split = np.linspace(min(X.humidity), max(X.humidity), 200)
humidity_splits = [X.humidity <= H for H in humidity_split]

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 [20]: 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
```



Das Attribut Temperatur eignet sich am besten.

0,5 P.

```
In [21]: 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')
```

3 Aufgabe 17

Wie sollten nicht-numerische Datentypen wie beispielsweise Strings vor der Analyse behandelt werden müssen? b) Kann es hilfreich sein Attribute zu normieren? Wenn ja, wieso? c) Wie kann mit Lücken in den Daten oder NaNs und Infs verfahren werden? d) Was ist beim Zusammenführen von Datensätzen zu beachten? e) Welche Attribute sollten vor dem Trainieren des Klassifizierers aus dem Datensatz entfernt werden. Wie kann dabei eine Reduktion redundanter Informationen erreicht werden? Was muss speziell bei simulationsbasierten Methoden berücksichtigt werden?

- a) Aus den Strings sollten vergleichbare Attribute generiert werden, z.B. die Länge oder die Anzahl an bestimmten Buchstaben. Falls die Strings nur eine Eigenschaft angeben, kann diese einer Zahl zugeordnet werden. ✓ 0,5 P.
- b) Dies kann z.B. für den kNN Algorithmus relevant sein, wie oben bereits erklärt. ✓ 0,5 P.
- c) Man kann diese Attribute oder den jeweiligen Datenpunkt einfach weglassen. Oder interpretieren, was z.B. Infs in dem Kontext bedeuten und dann entsprechend einen anderen Wert

Kölnwetter, Randrechte ✓ 0,5P.

zuweisen.

d) Die Datensätze müssen über die selben Attribute verfügen.

e) Attribute, die für alle Daten gleich sind enthalten z.B. keinerlei Information. Manche Daten stehen vielleicht in einem funktionellen Zusammenhang zueinander. Eine Reduktion der Attribute kann z.B. mit der PCA erreicht werden.

0,5P.

0,5P.

0,5P.

Was ist bei Konsolidationsbasierenden Methoden? ✓ -0,5P.

zuweisen.

0,5P.

0,5P.

0,5P.

a) $H(S)$. Fall im 4 Quadranten → S nicht

→ S nicht

$$H(S) = -\frac{9}{14} \log_2\left(\frac{9}{14}\right) - \frac{5}{14} \log_2\left(\frac{5}{14}\right) \approx 0.340286 \quad \checkmark \quad 0,5P.$$

$$H(W_{\text{skew}}) = -\frac{6}{8} \log_2\left(\frac{6}{8}\right) - \frac{2}{8} \log_2\left(\frac{2}{8}\right) = 0.812781 \quad \checkmark$$

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

→ 3 meint

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

$$\text{Gain}(W_{\text{skew}}) = 0.340286 - 0.81278 = 0.128008$$

$$\text{Gain}(W_{\text{skew}}) = 0.340286 - 1 = -0.659714$$

$$\text{Gesamt: } \text{Gain}(W_{\text{skew}}) = H(S) - \frac{6}{14} H(W_{\text{skew}}) - \frac{6}{14} H(W_{\text{skew}})$$

$$= 0.048127 \quad \checkmark$$

2P.

c) / d) siehe Tabelle

A16

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

$$\text{b) } P(W_1|F) = P(W_1|F) P(W_2|F) P(W_3|F) P(W_4|F) \quad \text{F}$$

$P(W_1) = P(W_1)P(W_2)P(W_3)P(W_4)$ denkt unabhängig
gesucht $P(ja)$ stark, hoch, halb, schwach

$$P(\text{stark})|ja = \frac{1}{3} \quad P(\text{hoch})|ja = \frac{1}{3} \quad P(\text{halb})|ja = \frac{1}{3}$$

$$P(\text{schwach})|ja = \frac{1}{3} \quad P(\text{schwach}) = \frac{1}{4} \quad P(\text{kalt}) = \frac{3}{14}$$

$$P(\text{schwach}) = \frac{3}{14} \quad P(\text{schwach}) = \frac{1}{4} \quad P(\text{kalt}) = \frac{3}{14}$$

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

$$P(W|F) = \frac{1}{3} \cdot \frac{1}{3} \cdot \frac{1}{3} = \frac{1}{243} \quad P(F) = \frac{9}{144}$$

$$P(W) = \frac{3}{4} \cdot \frac{1}{2} \cdot \frac{3}{2} \cdot \frac{3}{14} = \frac{27}{1344} \quad \text{f} - 10.$$

$$\Rightarrow P(F|W) = \frac{156}{229} \approx 0,27$$

\Rightarrow Die Wahrscheinlichkeit, dass Fünfball gespielt wird beträgt 27 %

$$\hookrightarrow P(\text{Kunstlich}) = 0$$

\rightarrow Wahrscheinlichkeit war direkt null

\rightarrow Beidein stattetum Pfeil (siehe Punkt W mögl.)

$$P(W_{\text{neut}}) = \frac{1}{5} \cdot \frac{4}{5} \cdot \frac{1}{5} \cdot \frac{1}{5} = \frac{1}{625}$$

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

$$P(W_{\text{neu}}) = \frac{5}{14}$$

$$P(W_{\text{neu}}|W_{\text{neg}}) = \frac{\frac{5}{14}}{\frac{625}{686}} \approx 0,976$$

$$\Rightarrow P(\text{gut W neg}) = 1 - P(W_{\text{neg}}) \approx 2,4 \%$$

andere Möglichkeiten:

$$P(\text{grau | schwarz \& weiß})$$

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

ich glaube das wolle mir das richtige Sagen
10.