Freie Universität Berlin

Prof. Dr. Agnès Voisard
Nicolas Lehmann

Datenbanksysteme, SoSe 18

Übung 04

TutorIn: Toni Draßdo
Tutorium 014

Eduard Beiline, Mark Niehues, Antoen Oehler

20. Mai 2018

Task 1: ER-Modellierung

Task 2: Relationales Modell

Task 3: Reverse Engineering

Task 4: Data Mining

1 - K-Means

```
Center for this Cluster: [ 6.66666667  9.66666667  1.66666667]
Contains:
[ 3 10  1]
[10 10  3]
[7 9 1]

Center for this Cluster: [ 7.33333333  3.66666667  1.66666667]
Contains:
[9 2 3]
[9 4 1]
[4 5 1]

Center for this Cluster: [ 5.   4.5  6.5]
Contains:
[6 6 8]
[4 3 5]
```

Abbildung 1: Ergebnis des K-Means Algorithmus

Listing 1: K-Means Implementierung

```python
#!/usr/bin/env python3
# coding: utf-8
import numpy as np
import sys

DATA = np.array([
    [3,9,9,10,6,7,4,4],
    [10,2,4,10,6,9,5,3],
    [1,3,1,3,8,1,1,5]
]).T


def expectation(data, centers):
    """
    Assigns datapoints to centers
    """
    clusters = [[] for _ in range(centers.shape[0])]

    for point in data:
            # Calculate distances from centers
            d = np.linalg.norm(centers - point, axis=1)

            # Find the index according to the lowest distance
            id_min = np.argmin(d)

            # Assing point to minimum distance
            clusters[id_min].append(point)

    for clt in clusters:
        if len(clt) == 0:
            print("Error: Empty cluster occured, please re-run the program.")
            sys.exit(1)
    return clusters


def minimization(clusters):
    """
    Computes new cluster means
    """
    centers = [np.mean(cls, axis=0) for cls in clusters]

    return np.vstack(centers)


def k_means(data, k, sigma):
    # Initialize with k random points
    centers = data[np.random.randint(data.shape[0], size=k)]

    dist = sigma + 1
    while dist > sigma:
        # Assign data points to centers
        clusters = expectation(data, centers)

        # Calculate new centers
        new_centers = minimization(clusters)

        # Calc maximum center movement
        dist = max(np.linalg.norm(centers - new_centers, axis=1))
        centers = new_centers

    return centers, clusters


def print_clusters(centers, clusters):
    for center, clst in zip(centers, clusters):
        print("Center for this Cluster: {}".format(center))
        print("Contains:")
```

```
        for _ in clst:
            print(_)
        print()


centers, clusters = k_means(DATA, k=3, sigma=3/4)
print_clusters(centers, clusters)
```

2 - Naive Bayes

1 - Wahrscheinlichkeit einer Grippe bei laufender Nase

Naive Bayes Formula:

$$P(C|x) = \frac{P(C)\,P(x|C)}{P(x)} \tag{1}$$

Aus Formel 1 folgt für die Wahrscheinlichkeit an einer Grippe zu leiden, bei laufender Nase:

$$P(Grippe|Nase) = \frac{P(Gripp)\,P(Nase|Grippe)}{P(Nase)} \tag{2}$$

wobei:

$$P(Nase) = 4/8 = 1/2$$
$$P(Grippe) = 1/2$$
$$P(Nase|Grippe) = 3/4$$

Durch einsetzen in Formel 2 erhält man:

$$P(Grippe|Nase) = 3/4$$

2 - Grippe, wenn X

Um die Frage zu beantwortet, ob jemand eher Grippe oder keine Grippe besitzt wird, um die Rechnung zu Vereinfachen der Quotient aus $P(Grippe|x)$ und $P(\neg Grippe|x)$ gebildet, dadurch kürzt sich die aufwändig zu berechnende Evidenz $P(x)$ heraus:

$$Q = \frac{P(Grippe|x)}{P(\neg Grippe|x)} = \frac{P(x|Grippe)\,P(Grippe)}{P(x|\neg Grippe)\,P(\neg Grippe)} \tag{3}$$

wobei:

$$
\begin{aligned}
&x = \{Schttelfrost, schwacheKopfschmerzen, Fieber\}\\
&P(Schttelfrost|Grippe) = 3/4\\
&P(Schttelfrost|\neg Grippe) = 1/2\\
&P(schwacheKopfschmerzen|Grippe) = 1/4\\
&P(schwacheKopfschmerzen|\neg Grippe) = 1/4\\
&P(Fieber|Grippe) = 1/2\\
&P(Fieber|\neg Grippe) = 1/2\\
&P(Grippe) = P(\neg Grippe) = 1/2
\end{aligned}
$$

Unter Annahme der (hinreichenden) Unabhängigkeit der Variablen, gilt $P(x_1, x_2|C) = P(x_1|C)\,P(x_2|C)$. Daraus ergibt sich schließlich:

$$
Q = \frac{\frac{3}{4}\,\frac{1}{4}\,\frac{1}{2}}{\frac{1}{2}\,\frac{1}{4}\,\frac{1}{2}} = 3/2
$$

Aus der $Q > 1$ folgt, dass der Patient wahrscheinlicher Grippe hat als keine.

3 - Apriori

Listing 2: Ergebnis des Apriori Algorithmus unter Zuhilfename der Implementierung aus [1].

```
item: ('F',) , 0.500
item: ('A',) , 0.500
item: ('C',) , 0.500
item: ('D', 'F') , 0.500
item: ('A', 'E') , 0.500
item: ('C', 'B') , 0.500
item: ('C', 'D') , 0.500
item: ('A', 'B') , 0.500
item: ('E', 'F') , 0.500
item: ('C', 'B', 'D') , 0.500
item: ('E', 'D', 'F') , 0.500
item: ('A', 'B', 'E') , 0.500
item: ('B', 'E', 'D') , 0.500
item: ('B', 'D') , 0.667
item: ('B', 'E') , 0.667
item: ('E', 'D') , 0.667
item: ('D',) , 0.833
item: ('B',) , 0.833
item: ('E',) , 0.833

------------------------ RULES:
Rule: ('B', 'D') ==> ('C',) , 0.750
Rule: ('E', 'D') ==> ('F',) , 0.750
Rule: ('B', 'E') ==> ('A',) , 0.750
Rule: ('B', 'E') ==> ('D',) , 0.750
Rule: ('B', 'D') ==> ('E',) , 0.750
Rule: ('E', 'D') ==> ('B',) , 0.750
Rule: ('B',) ==> ('D',) , 0.800
Rule: ('D',) ==> ('B',) , 0.800
Rule: ('B',) ==> ('E',) , 0.800
Rule: ('E',) ==> ('B',) , 0.800
Rule: ('E',) ==> ('D',) , 0.800
Rule: ('D',) ==> ('E',) , 0.800
Rule: ('F',) ==> ('D',) , 1.000
Rule: ('A',) ==> ('E',) , 1.000
Rule: ('C',) ==> ('B',) , 1.000
```

```
Rule: ('C',) ==> ('D',) , 1.000
Rule: ('A',) ==> ('B',) , 1.000
Rule: ('F',) ==> ('E',) , 1.000
Rule: ('C',) ==> ('B', 'D') , 1.000
Rule: ('C', 'B') ==> ('D',) , 1.000
Rule: ('C', 'D') ==> ('B',) , 1.000
Rule: ('F',) ==> ('E', 'D') , 1.000
Rule: ('E', 'F') ==> ('D',) , 1.000
Rule: ('D', 'F') ==> ('E',) , 1.000
Rule: ('A',) ==> ('B', 'E') , 1.000
Rule: ('A', 'B') ==> ('E',) , 1.000
Rule: ('A', 'E') ==> ('B',) , 1.000
```

4 - Lineare Regression

Leider ist bei uns erst eine Korrektur eingetragen, deswegen können wir keine lineare Regression durchführen (unterbestimmtes System).

Literatur

[1] asaini. *Python Implementation of Apriori Algorithm*. https://github.com/asaini/Apriori.git. 2017.