Datenbanksysteme SS17
Projektdokumentation

Freie Universität Berlin
Fachbereich Informatik

Ralitsa Dineva, Nedelcho Petrov, Noah Schick

10. Juli 2017

Einleitung

Dieses Projekt findet statt im Rahmen des Moduls Datenbanksysteme. Das Ziel ist eine Web-Anwendung anzufertigen, die die Daten aus dem Datensatz american-election-tweets visualisiert und Abfragen beantworten kann. Der Datensatz enthält Information über die tweets von der Präsidentschaftskandidaten in der USA in 2016. Mit Hilfe dieser Daten wird mit SQL eine Datenbank erstellt. Danach kann durch verschiedene Operationen die abgefragte Information abgeleitet werden. Die Web-Anwendung wird benutzerfreundlich und leicht zu bedienen sein und die Abfrageergebnisse schön und klar dargestellt.

Das Projektentwicklung wird in drei Iterationen verlaufen:

1. **Modellierung**

   1.2 Explorative Datenanalyse des Datensatzes

   1.3 ER-Modell

   1.4 Relationales Modell

   1.5 Erstellung von Datenbank

2. **Datenimport**

   2.1 Datenbankschema erstellen

   2.2 Datenbereinigung

   2.3 Datenimport

   2.4 Webserver

3. **Data Mining und Visualisierung**

   3.1 Data Mining: Clustering

      3.1.1 Metrik für die Ähnlichkeit von Hashtags

      3.1.2 Cluster finden

      3.1.3 Visualisation

   3.2 Datenvisualisierung

      3.2.1 Hashtag Netzwerk

      3.2.2 Hashtag Netzwerk mit Ähnlichkeit von Hashtags

      3.2.3 Häufigkeit des Auftretens aller Hashtags

      3.2.4 Häufigkeit des Auftretens von bestimmten Hashtag

Link zu Repository:
https://github.com/ralitsadineva/DBSProjekt/blob/master

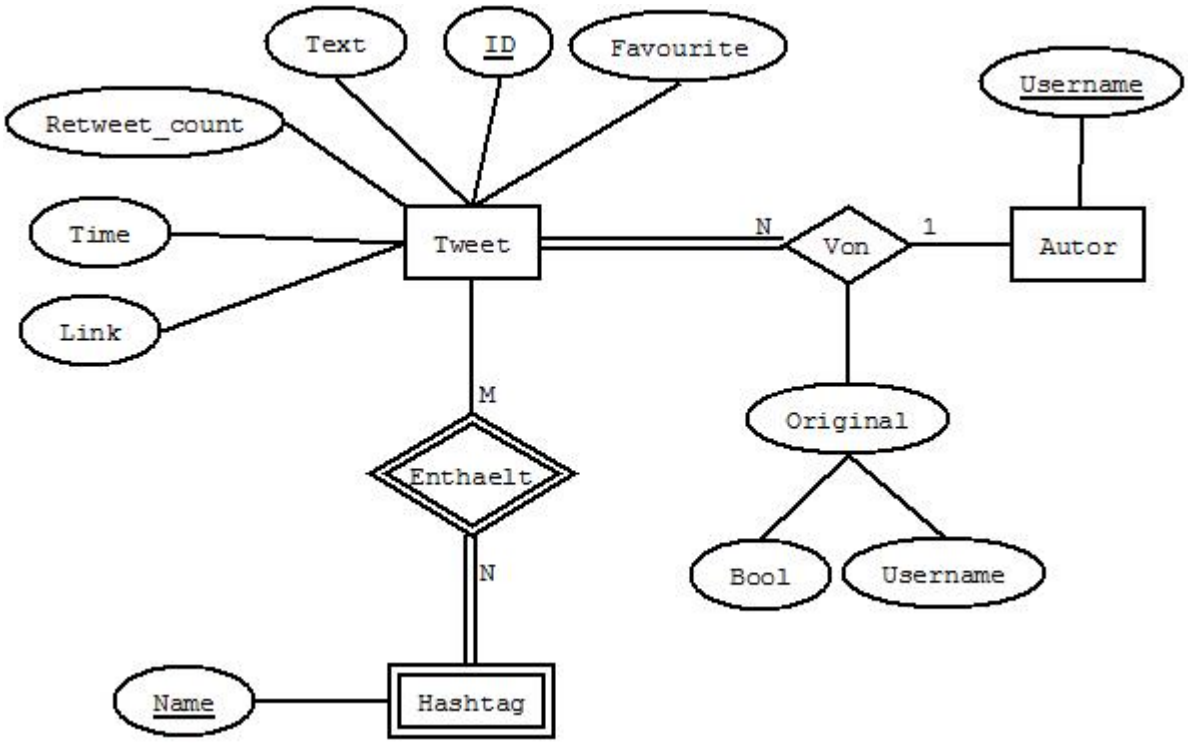
Unser Team

Auf dem Projekt arbeitet eine Gruppe von drei Studenten: Ralitsa Dineva, Nedelcho Petrov und Gulnaz Kazakbaeva. Sie sind alle Informatik Studenten an der FU Berlin. Gesammelt von verschiedenen Ecken der Erde um zusammen das Projekt zu erstellen, sie versuchen sich tief in der Lehre von Datenbanksystemen zu vertiefen und werden ihre Herzen und Schwitze bis den letzten Moment für das Erfolg geben.

1.2 Explorative Datenanalyse des Datensatzes

Der Datensatz ist eine Darstellung aller Tweets von Trump und Clinton vor den Präsidentschaftswahl in USA 2016. Die Hauptinformation in der Tabelle sind alle verschiedene Tweets. Zu jeden Tweet steht von wem und wann er gepostet war (Trump/Clinton). Falls er ein Retweet war, dann noch zusätzlich der Username des Originalautors. Im weiteren zeigen die üblichen Spalten ob der Tweet ein Zitat war, wie viele Benutzer den retweetet haben, wie viele ihn als favorite markiert haben und seine Internet Adresse, bzw mit welcher Gerät das getweetert wurde.
Die Information in den Spalten 'is quote status' und 'truncated' ist unnötig für das Ziel des Projekts und wird in seine weitere Entwicklung nicht betrachtet.

1.3 ER-Modell



Erklärung des Diagrams und der wichtigsten Entscheidungen:

Das Diagramm besteht aus den drei Entitätstypen $Tweet$, $Autor$ und $Hashtag$ (schwach) und die zwei Relationen Von und $Enthaelt$.

Die Entitätstypen:

- $Tweet$: steht für alle Tweets in dem Datensatz mit Schlüsselattribut ID und noch:
  - $Text$: Inhalt des Tweets.
  - $Time$: Die Zeit der Erscheinung.
  - $Retweet_Count$: Anzahl von Retweets.
  - $Favourite$: Anzahl von Likes.
  - $Link$: URL Link des Tweets
- $Autor$ steht für Hillary Clinton und realDonaldTrump. Der Schlüsselattribut ist $Username$
- $Hashtag$ steht für alle Hashtags mit Schlüsselattribut $Name$ die Name des Hashtags. Das ist einen schwachen Entytatstyp, da er von $Tweet$ durch einen Algorithmus* erzeugt wird.

  *Der Algorithmus liest den Text des Tweets durch und macht aus jedem Wort das mit '#' anfängt eine Instanz.

Die Relationen:

- Von ist Relation zwischen $Tweet$ (1:1) und $Autor$ (1:N) die zeigt wer den Tweet getweetet hat. Um solche Tweets, die originell von Trump oder Clinton geschrieben sind nachvollziehen zu können, hat Von den Attribut $Original$. Er ist ein Tupel (Bool, Username), der falls der Tweet von jemand anderen geschrieben ist die Form (False, < Username des originellen Autors>) hat und sonst (True, NULL).
- $Enthaelt$ zeigt welcher Tweet welche Hashtags in sich ethält.

1.4 Relationales Modell

Entitäten:

- $TWEET$(ID, Text, Time, Favourite, Retweet_count, Link)
- $HASHTAG$($Name$)
- $AUTOR$($Username$)

Relationen:

- VON(TID, Username, Original)
- $ENTHAELT$(TID, HName)

TID ist in beide Fällen der Fremdschlüssel von $Tweet$ und $HName$ ist von $Hashtag$.

Begründung des Funktionalitäts:

- Welche 'Tweets' waren am wichtigsten?
  Man hat die favourite_count und retweet_count als Attributen und kann dann Operationen wie MAX und AVG und eigene Kriterien nutzen um die wichtigsten Tweets zu erzeugen.
- Welche Hashtags wurden am meisten verwendet?
  Es wird für jede Instanz von $Hashtag$ die Anzahl von Tweets in dem er vorkommt genommen:

```
SELECT Name, Count(*)
FROM ENTHAELT
WHERE
GROUPBY Name
```

  Man konnte auch den favourite_count jedes einzelnes Tweet betrachten und dadurch noch besser auswählen.

- Wann traten insgesammt am meisten Hashtags auf?
  Mittels:
  $$\Pi_{Name,Time}((\text{Enthaelt}) \underset{\{\text{TID = ID}\}}{\bowtie} (\text{Tweet}))$$
  entsteht eine Relation mit der Zeit der Erscheinung von jeden Hashtag. Davon kann berechnet werden wann traten am häufigsten Hashtags auf.

- Paaren (#A, #B) von Hashtags;
  Welche Hashtags traten besonders häufig gemeinsam an?;
  Wie hat sich die Häufigkeit der Verwendung eines speziellen Hashtags über die Zeit entwickelt?

  ```
  SELECT A.Name, B.Name, TID
  FROM ENTHAELT.A, ENTHAELT.B
  WHERE A.TID = B.TID
  ```

  Davon entstehen alle Paare von Hashtags und den Tweet in dem sie gemeinsam vorkommen. Der Attribut $Time$ kann auch genomment werden um Häufigkeiten zu messen. Ähnlich kann jeden Hashtag einzeln bertachtet werden.

1.5 Erstellung von Datenbank

Siehe den Screenschots.

2.1 Datenbankschema erstellen

Relationeles model (siehe 1.4) in PSQL:

```
DROP TABLE IF EXISTS VON;
DROP TABLE IF EXISTS ENTHAELT;
DROP TABLE IF EXISTS HASHTAG;
DROP TABLE IF EXISTS AUTOR;
DROP TABLE IF EXISTS TWEET;

CREATE TABLE TWEET(
        ID SERIAL PRIMARY KEY,
        Text text NOT NULL,
        Time date,
        Favourite integer,
        Retweet_count integer,
        Link text);

CREATE TABLE HASHTAG(
        Name text PRIMARY KEY    );

CREATE TABLE AUTOR(
        Username text PRIMARY KEY);

CREATE TABLE VON(
        TID integer REFERENCES TWEET(ID),
        Username text REFERENCES AUTOR(Username),
        Original text,
        PRIMARY KEY(TID, Username));

CREATE TABLE ENTHAELT(
        TID integer REFERENCES TWEET(ID),
        HName text REFERENCES HASHTAG(Name),
        PRIMARY KEY(TID, HName));
```

Link zu Repository: `https://github.com/ralitsadineva/DBSProjekt/blob/master/create_tables.sql`

2.2 Datenbereiningung

Operationen die auf den Daten angewendet sind:

- Hashtags in den Tweets finden und rausnehmen.

- Das 'T' zwischen Den und Uhrzeit mit Leerzeichen erstezten.

- Das Link zu dem Tweet von dem Text rausnehmen.

Code:

```
import csv

#Openning the data file:
csvfile = open('american-election-tweets.csv')
creader = csv.DictReader(csvfile, delimiter=';')

read_file = [row for row in creader]

#Finding hashtags:
def extract_hash_tags(s):
    s = s.replace("# ", "#")
    return set(part[1:] for part in s.split() if part.startswith('#'))

#Extracting hashtags:
def get_all_hashtags(creader):
    hashtags = set()
    for row in creader:
        hashtags.update(extract_hash_tags(row['text']))
    return hashtags

#Formats the date and time from YY-MM-DD"T"HH-MM-SS
#to YY-MM-DD HH-MM-SS:
def format_timestamp(timestamp):
    return timestamp.replace("T", " ")
```

```
#Separates the text of a tweet from the link at the end of it:
def extract_links(text):
    links = [part for part in text.split() if part.startswith('https')]
    if links == []:
        return []
    return links[len(links) - 1]
```

Link zu Repository: `https://github.com/ralitsadineva/DBSProjekt/blob/master/read_and_correct.py`

2.3 Datenimport

Code der den Datensatz eignet und dann in den Datenbank 'election' importiert:

```
import psycopg2
from read_and_correct import *

db = "election"
username = "postgres"
passwort = "ENTER YOUR PASSWORD"
host_ = "localhost"
port_ = "5432"
conn = psycopg2.connect(database = db, user = username,...
...password = passwort, host = host_, port = port_)
print("Opened database successfully")

cur = conn.cursor()

def populate_tweets():
    print("Populating tweets...")
    for row in read_file:
        link = extract_links(row['text'])
        if link != []:
            url = link
        else:
            url = row['source_url']
        cur.execute('INSERT INTO TWEET (Text, Time, Favourite,...
        ...Retweet_count, Link) VALUES (%s, %s, %s, %s, %s)',...
        ...(row['text'], format_timestamp(row['time']),...
        ...row['favorite_count'], row['retweet_count'], url))
    print("Finished populating tweets!")

def populate_hashtags():
    print("Populating hashtags...")
    hashtags = get_all_hashtags(read_file)
    for hashtag in hashtags:
        if len(hashtag) > 0:
```

```
                cur.execute("INSERT INTO HASHTAG VALUES (%s)", (hashtag, ))
        print("Finished populating hashtags!")


def populate_autor():
    print("Populating autor...")
    authors = set()
    for row in read_file:
        authors.add(row['handle'])

    for author in authors:
        cur.execute("INSERT INTO AUTOR VALUES(%s)", (author, ))
    print("Finished populating autor!")

def populate_von():
    print("Populating von...")
    cur.execute("SELECT * FROM TWEET")
    rows = cur.fetchall()
    for row in rows:
        id = row[0]
        text = row[1]
        time = row[2]
        favourite = row[3]
        retweet_count = row[4]
        for row_file in read_file:
            if text == row_file['text'] and str(time) == row_file['time
                if row_file['is_retweet'] == 'True':
                    og =  row_file['original_author']
                else:
                    og = row_file['handle']
                cur.execute("INSERT INTO VON VALUES(%s, %s, %s)", (id,
                break
    print("Finished populating von!")

def populate_enthaelt():
    print("Populating enthaelt...")
    cur.execute("SELECT * FROM TWEET")
    rows_tweet = cur.fetchall()
```

```
    cur.execute("SELECT * FROM HASHTAG")
    rows_hashtag = cur.fetchall()
    for r_tweet in rows_tweet:
        id = r_tweet[0]
        text = r_tweet[1]
        for r_hash in rows_hashtag:
            if "#" + r_hash[0] in text.replace("# ", "#"):
                cur.execute("INSERT INTO ENTHAELT VALUES(%s, %s)", (id,
    print("Finished populating enthaelt!")


populate_hashtags()
populate_autor()
populate_tweets()
populate_von()
populate_enthaelt()
conn.commit()
cur.close()
conn.close()
```

Link zu Repository: `https://github.com/ralitsadineva/DBSProjekt/blob/master/populate_db.py`

2.4 Webserver

Es wurde einen Apache Tomcat installiert und cofiguriert nach diesem Tutroial: `https://www.howtoforge.com/tutorial/how-to-install-apache-tomcat-8-5-on-ubuntu-`

3.1 Data Mining: Clustering

3.1.1 Metrik für die Ähnlichkeit von Hashtags

Als Metrik für die Ähnlichkeit von Hashtags haben wir die Levenshtein-Distanz ausgewählt. Die Distanz zwischen zwei Zeichenketten ist die Anzahl von Einfüge-, Lösch- und Ersetz-Operationen die auf der ersten ausgeführt werden müssen, sodass sie in der zweiten umgewandelt ist.

3.1.2 Cluster finden

Im folgenden Code ist der Algorithm für die Distanz und das finden von Clustern implementiert:

cluster.py:

```
import psycopg2
import random

db = "election"
username = "postgres"
passwort = "ENTER YOUR PASSWORD"
host_ = "localhost"
port_ = "5432"
conn = psycopg2.connect(database = db, user = username,...
                        ... password = passwort, host = host_,...
                        ... port = port_)
print("Opened database successfully")

cur = conn.cursor()

def hashtags_list():
    cur.execute("SELECT * FROM hashtag")
    hashtags = []
    data = cur.fetchall()
    for i in data:
        hashtags.append(i[0])
    return hashtags
```

```
hashtags = hashtags_list()

conn.commit()
cur.close()
conn.close()

def levenshtein(s1, s2):
    if len(s1) < len(s2):
        return levenshtein(s2, s1)

    # len(s1) >= len(s2)
    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row

    return previous_row[-1]

distance_table = [[0 for _ in range(len(hashtags))] for _ in...
...range(len(hashtags))]
index_dict = dict()

def init_table():
    for i in range(len(hashtags)):
        index_dict[hashtags[i]] = i
        for j in range(len(hashtags)):
            distance_table[i][j] = levenshtein(hashtags[i], hashtags[j])
```

```
def initialize_centers(k):
    hashtags_copy = hashtags[:]
    random.shuffle(hashtags_copy)
    centers = hashtags_copy[0:k]
    return centers

def get_distance(h1, h2):
    index1 = index_dict[h1]
    index2 = index_dict[h2]
    return distance_table[index1][index2]

def find_closest_center(h_tag, centers):
    center = min(centers, key=lambda c: get_distance(h_tag, c))
    return centers.index(center)

def assign_hashtags_to_clusters(hashtags, centers):
    clusters = [[] for _ in range(len(centers))]
    for h_tag in hashtags:
        c = find_closest_center(h_tag, centers)
        clusters[c].append(h_tag)
    return clusters

def find_cost(clusters, centers):
    sum_total = 0
    for index, cluster in enumerate(clusters):
        sum_cluster = 0
        center = centers[index]
        for element in cluster:
            sum_cluster += get_distance(center, element)
        sum_total += sum_cluster
    return sum_total
```

```
def clustering(hashtags):
    init_table()
    k = int(input("Enter value for k: "))
    centers = initialize_centers(k)
    clusters = assign_hashtags_to_clusters(hashtags, centers)
    cost = find_cost(clusters, centers)
    center_index = 0
    centers_set = set(centers)
    while center_index < k:
        for hashtag in hashtags:
            cost_new = 0
            if hashtag not in centers_set:
                centers_new = centers[:]
                centers_new[center_index] = hashtag
                clusters_new = assign_hashtags_to_clusters(...
                ...hashtags, centers_new)
                cost_new = find_cost(clusters_new, centers_new)
                if cost_new < cost:
                    centers = centers_new
                    clusters = clusters_new
                    cost = cost_new
                    centers_set = set(centers)
        center_index += 1
    return clusters

if __name__ == "__main__":
    clusters = clustering(hashtags)
    for i in range(len(clusters)):
        print("Cluster " + str(i) + ": " + str(clusters[i]))
```

3.1.3 Visualisation

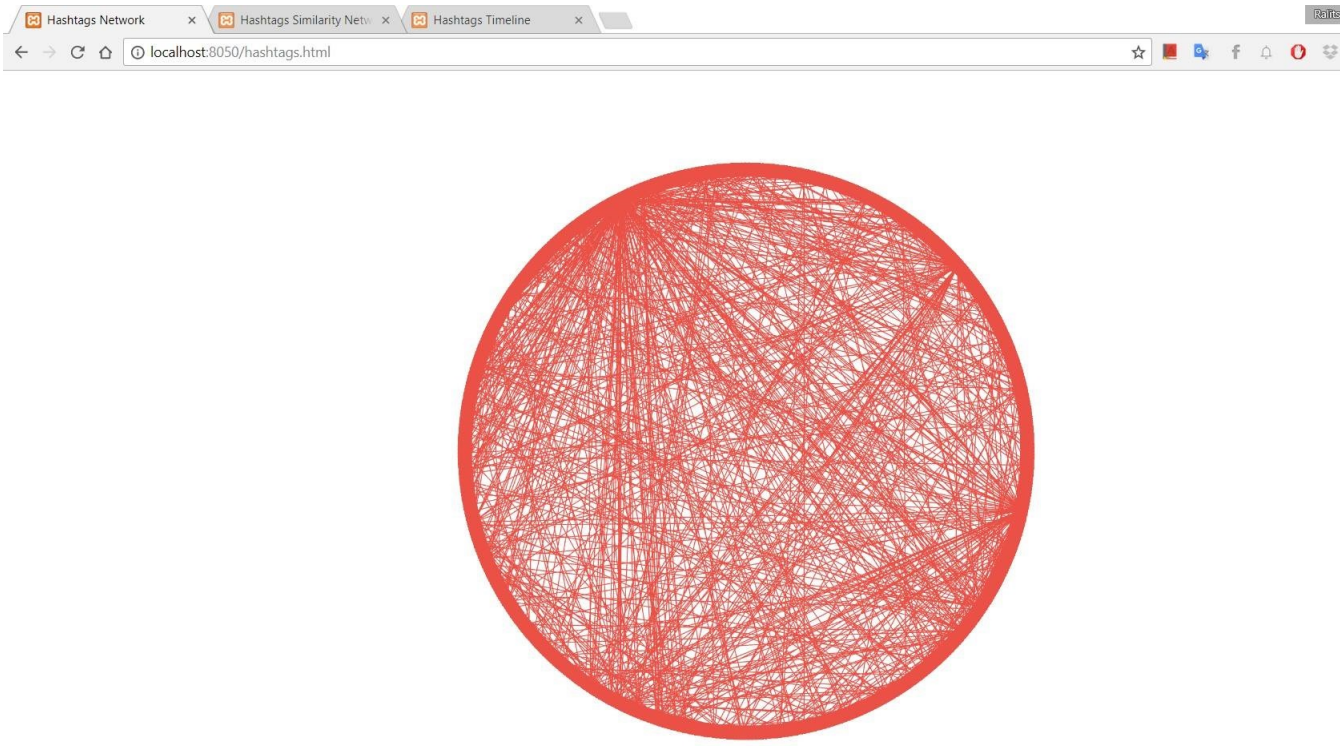
Die Cluster werden nach ihrer Erstellung Ausgedrückt.

3.2 Datenvisualisierung

3.2.1 Hashtag Netzwerk

Bild 1:

Bild 2:



create_ json.py:

```
import json
import psycopg2
import math

db = "election"
username = "postgres"
passwort = "ENTER YOUR PASSWORD"
host_ = "localhost"
port_ = "5432"
conn = psycopg2.connect(database = db, user = username,...
                        ... password = passwort, host = host_, port = port_)
print("Opened database successfully")

cur = conn.cursor()
```

```
def get_data_nodes():
    cur.execute("SELECT * FROM hashtag")
    nodes = []
    data = cur.fetchall()
    for i in data:
        nodes.append(i[0])
    return nodes

def get_data_edges():
    cur.execute("SELECT h1.hname, h2.hname FROM enthaelt h1 JOIN...
    ... enthaelt h2 on h1.tid = h2.tid WHERE h1.hname != h2.hname")
    edges = []
    data = cur.fetchall()
    for i in data:
        edges.append(i)
    return edges

data_nodes = get_data_nodes()
data_edges = get_data_edges()

conn.commit()
cur.close()
conn.close()

data_n = []
n = 0
for i in data_nodes:
    data = {
        "id": "n" + str(n),
        "label": i,
        "x": math.cos(math.pi * 2 * n / len(data_nodes)),
        "y": math.sin(math.pi * 2 * n / len(data_nodes)),
        "size": 1
    }
    data_n.append(data)
    n += 1
```

```
data_e = []
e = 0
source_n = ""
target_n = ""
for i in data_edges:
    for item in data_n:
        if item["label"] == i[0]:
            source_n = item["id"]
        if item["label"] == i[1]:
            target_n = item["id"]
    data = {
        "id": "e" + str(e),
        "source": source_n,
        "target": target_n
    }
    data_e.append(data)
    e += 1

data = {
    "nodes": data_n,
    "edges": data_e
}

with open('data.json', 'w') as f:
     json.dump(data, f)
```

hashtags.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hashtags Network</title>
  <style type="text/css">
    #container {
      position: absolute;
      width: 100%;
      height: 100%;
      margin: auto;
    }
  </style>
</head>
<body>
  <div id="container"></div>
  <script src="./sigma.min.js"></script>
  <script src="sigma.parsers.json.min.js"></script>
  <script>
    sigma.parsers.json('data.json', {
      container: 'container',
      settings: {
        defaultNodeColor: '#ec5148'
      }
    });
  </script>
</body>
</html>
```

3.2.2 Hashtag Netzwerk mit Ähnlichkeit von Hashtags

Bild 1:

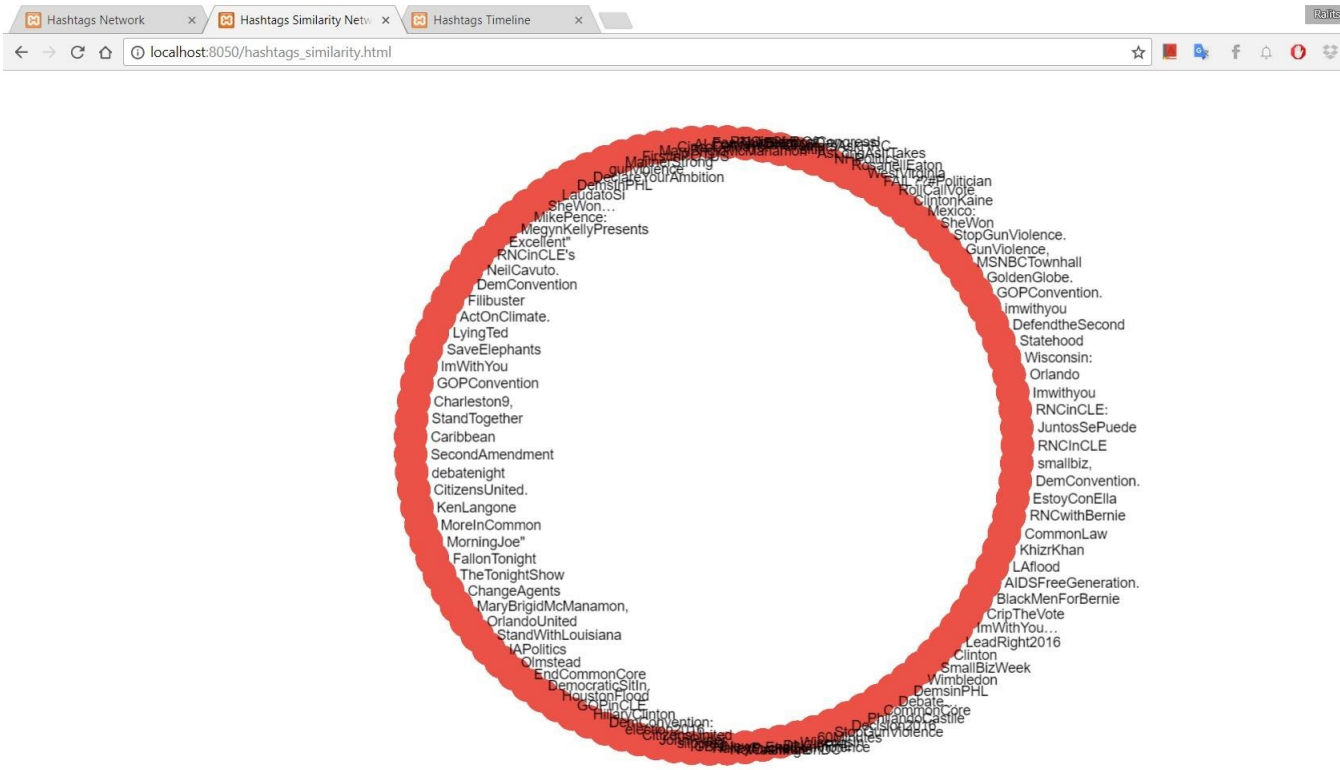
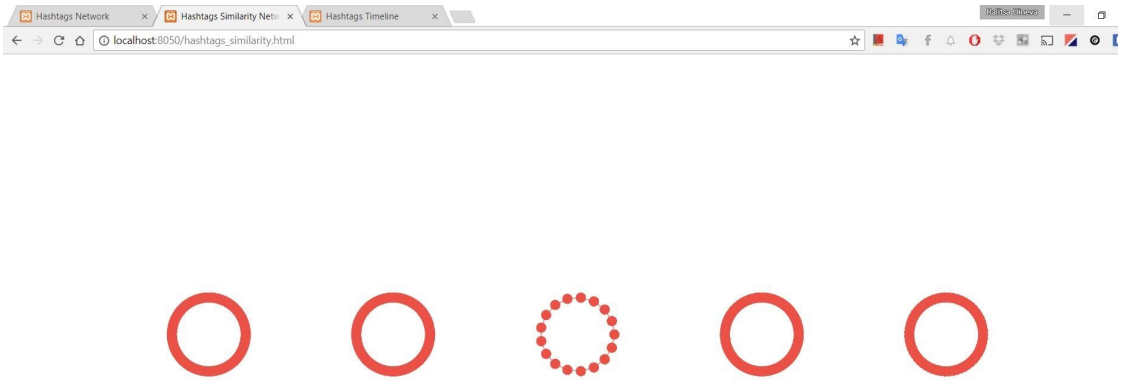


Bild 2:


Hashtags Network
Hashtags Similarity Netw
Hashtags Timeline
localhost:8050/hashtags_similarity.html

create_json_sim.py:

```
import json
import math
from clustering import *

clusters = clustering(hashtags)

data_n = []
n = 0
k = 0
l = 0
for cluster in clusters:
    k += 1
    l = 0
    for hashtag in cluster:
        data = {
            "id": "n" + str(n),
            "label": hashtag,
            "x": math.cos(math.pi * 2 * l / len(cluster)) + k * 5,
            "y": math.sin(math.pi * 2 * l / len(cluster)),
            "size": 1
        }
        data_n.append(data)
        n += 1
        l += 1
```

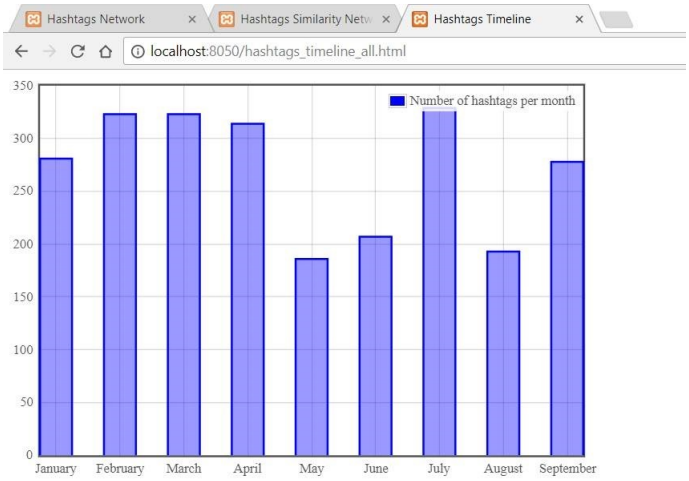
```
data_e = []
e = 0
source_n = ""
target_n = ""
for cluster in clusters:
    for i in range(len(cluster)):
        for item in data_n:
            if item["label"] == cluster[i]:
                source_n = item["id"]
            if i == len(cluster) - 1:
                if item["label"] == cluster[0]:
                    target_n = item["id"]
            else:
                if item["label"] == cluster[i+1]:
                    target_n = item["id"]
        data = {
        "id": "e" + str(e),
        "source": source_n,
        "target": target_n
        }
        data_e.append(data)
        e += 1

data = {
    "nodes": data_n,
    "edges": data_e
}

with open('data_sim.json', 'w') as f:
     json.dump(data, f)
```

hashtags_ similarity.html:

```
<!DOCTYPE html>
<html>
<head>
  <title>Hashtags Similarity Network</title>
  <style type="text/css">
    #container {
      position: absolute;
      width: 100%;
      height: 100%;
      margin: auto;
    }
  </style>
</head>
<body>
  <div id="container"></div>
  <script src="./sigma.min.js"></script>
  <script src="sigma.parsers.json.min.js"></script>
  <script>
    sigma.parsers.json('data_sim.json', {
      container: 'container',
      settings: {
        defaultNodeColor: '#ec5148'
      }
    });
  </script>
</body>
</html>
```

3.2.3 Häufigkeit des Auftretens aller Hashtags

Bild 1:



create_ file.py:

```
import psycopg2

db = "election"
username = "postgres"
passwort = "ENTER YOUR PASSWORD"
host_ = "localhost"
port_ = "5432"
conn = psycopg2.connect(database = db, user = username,
                        password = passwort,
                        host = host_, port = port_)
print("Opened database successfully")

cur = conn.cursor()

def get_data():
    cur.execute("SELECT t1.time FROM enthaelt h1...
    ... JOIN tweet t1 on h1.tid = t1.id")
    file = open("data_diagramm.txt", "w")
    data = cur.fetchall()
    for i in data:
        file.write(str(i[0]) + "\n")
    file.close()

get_data()

conn.commit()
cur.close()
conn.close()
```

hashtags_ timeline_ all.html:

```
<!DOCTYPE html>
<html>
<head>
        <meta charset="utf-8"></meta>
        <title>Hashtags Timeline</title>
        <script src="jquery-3.2.1.min.js" type='text/javascript'></script>
        <script type="text/javascript" src="jquery.flot.js"></script>
</head>

<body>
        <div id="barchart" style="width:600px;height:400px;"></div>
        <script>
                var drawGraphs = function(content) {
                        var lines = content.split("\n");
                        var data = []
            var hashtags = [0, 0, 0, 0, 0, 0, 0, 0, 0]
                        for (var i = 0; i < lines.length - 1; i++) {
            //length - 1, because the last line is empty
            var d = new Date(lines[i]);
            var m = d.getMonth();
            hashtags[m] += 1
            }
            for (var i = 0; i < 9; i++) {
                data.push([i, hashtags[i]]);
                        }
            var months = [
            [0, "January"], [1, "February"], [2, "March"],
            [3, "April"], [4, "May"], [5, "June"],
            [6, "July"], [7, "August"], [8, "September"]
            ];

                        var barData = [{
                                label: "Number of hashtags per month",
                                data: data,
                                color: "blue"
                        }];
                        var barOptions = {
                                series: {
                                        bars: {
```

```
                                                show: true
                                        }
                                },
                                bars: {
                                        align: "center",
                                        barWidth: 0.5
                                },
                        xaxis: {
                            axisLabel: "Month",
                            ticks: months
                        },
                        yaxis: {
                            axisLabel: "Number of hashtags"
                        }
                                }
                                $.plot($("#barchart"), barData, barOptions);
                        }
                        $.ajax({
                                url: "data_diagramm.txt",
                                async: true,
                                success: drawGraphs
                        });
                </script>
</body>
</html>
```

3.2.4 Häufigkeit des Auftretens bestimmtes Hashtags

Nicht implementiert.