



Hochschule **RheinMain**  
University of Applied Sciences  
Wiesbaden Rüsselsheim

# PRIVACY RANKING

## Wahlprojekt

Letztes Update: 8. August 2017

Max Mustermann

Studienbereich Informatik  
Hochschule RheinMain



# GLIEDERUNG

1. Einführung or whatever
2. Unsere App
3. Webservice
4. Wie kommen wir an die Daten?
5. Datenbank
6. Kategorisierung und Bewertung der Apps

EINFÜHRUNG OR WHATEVER

# Alles was wir am Anfang erzählen wollen

UNSERE APP

## Unsere App halt

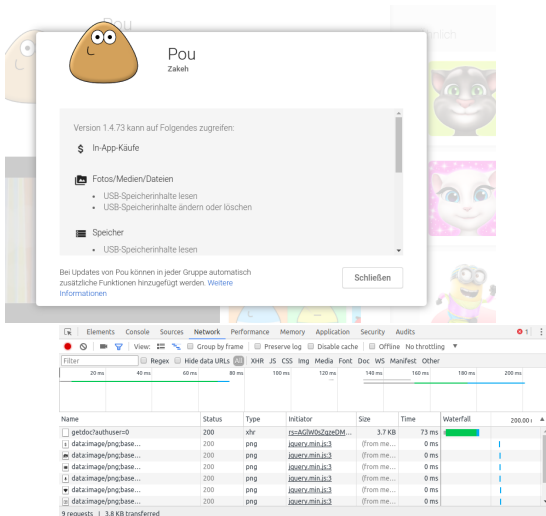
WEBSERVICE

# Webservice



WIE KOMMEN WIR AN DIE DATEN?

# WEBSITE GOOGLE PLAYSTORE



The image shows a screenshot of the Pou app interface with a permissions dialog box open. The dialog box is titled "Pou" and "Zakoh". It lists permissions that Version 1.4.73 can access:

- In-App-Käufe**
- Fotos/Medien/Dateien**
  - USB-Speicherinhalte lesen
  - USB-Speicherinhalte ändern oder löschen
- Speicher**
  - USB-Speicherinhalte lesen

At the bottom of the dialog, it says: "Bei Updates von Pou können in jeder Gruppe automatisch zusätzliche Funktionen hinzugefügt werden. [Weitere Informationen](#)". There is a "Schließen" button.

Below the app interface, the Chrome DevTools Network tab is visible. It shows a list of requests with the following columns: Name, Status, Type, Initiator, Size, Time, and Waterfall. The requests are as follows:

Name	Status	Type	Initiator	Size	Time	Waterfall
getdoc/authuser=0	200	xhr	rs=AGiWbZaZeDM...	3.7 KB	73 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	
dataimage/png/base...	200	png	jwercv.min.js	(from me...)	0 ms	

9 requests | 3.8 KB transferred

## SCRAPING DER DATEN

- ▶ Zugriff auf den Webservice von Google
- ▶ <https://play.google.com/store/xhr/getdoc?authuser=0>
- ▶ POST (ids=app\_id, xhr=1)

```
[["gdar",1,["me.pou.app","me.pou.app",1,3,
"/store/apps/details?id\u003dme.pou.app",
"/store/apps/details?id\u003dme.pou.app",
"https://play.google.com/store/apps/details
?id\u003dme.pou.app","https://market.android
.com/details?id\u003dme.pou.app","Pou",...
```

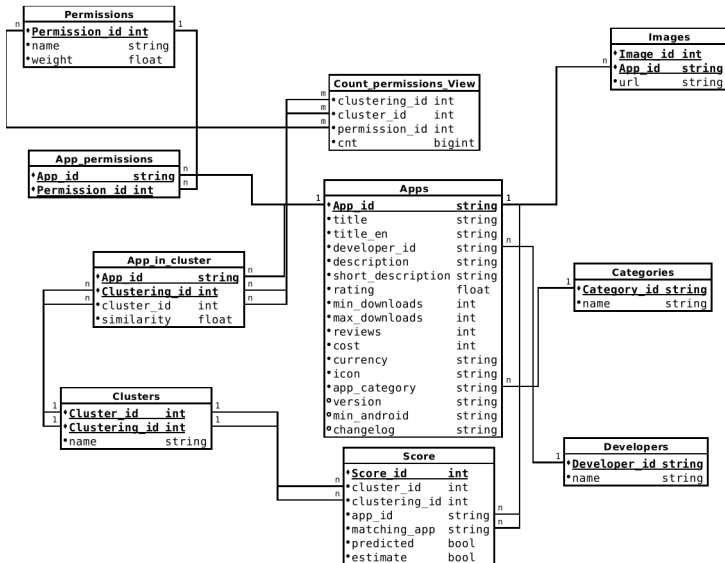
# EXTRAHIEREN DER DATEN

- ▶ Schreiben eines Wrappers in Python
- ▶ Lokalisieren der nötigen Informationen

```
def extract_title(data):  
    return _remove_emojis(data[0][2][0][8])  
  
def extract_description(data):  
    return _remove_emojis(data[0][2][0][9])  
  
def extract_rating(data):  
    return data[0][2][0][23]
```

DATENBANK

# MARIADB DATENBANK



# STORED PROCEDURE

```
DELIMITER $$
DROP FUNCTION IF EXISTS countAppsInCluster$$
CREATE FUNCTION countAppsInCluster( c_id INT,
    cing_id INT )
RETURNS INT DETERMINISTIC
READS SQL DATA
BEGIN
    DECLARE cnt INT;
    SELECT count(App_id) INTO cnt FROM
        App_in_cluster WHERE cluster_id = c_id
        AND Clustering_id = cing_id;
    RETURN cnt;
END$$
DELIMITER ;
```

⇒ Schlechte Idee, imperformant!

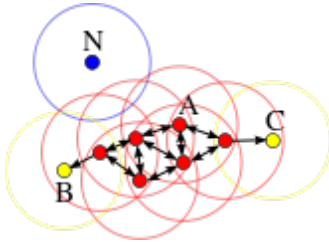
# KATEGORISIERUNG UND BEWERTUNG DER APPS



# DATAMINING

- ▶ Kategorisierung mithilfe von Clustering
- ▶ Auswahl zwischen den einzelnen Algorithmen
  - ▶ K-Means
    - ▶ Anzahl Cluster muss bekannt sein
  - ▶ Affinity propagation
    - ▶ Terminiert nicht
  - ▶ Mean-Shift
    - ▶ Terminiert nicht
  - ▶ Ward hierarchical clustering
    - ▶ Terminiert nicht
  - ▶ DBSCAN
    - ▶ Rauschen

# DBSCAN



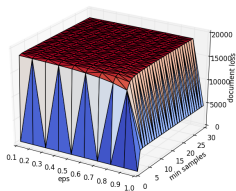
Quelle: Wikipedia

- ▶ Density-based spatial clustering of applications with noise
- ▶ Abstand (Epsilon) muss gut gewählt werden

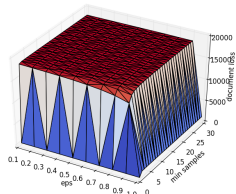
# TF-IDF

- ▶ Clustering-Algorithmen funktionieren nur mit numerischen Werten
- ▶ Text frequenz
  - ▶ Je häufiger Wort in Text enthalten  $\Rightarrow$  bedeutend
  - ▶ Wert für *min-df* muss gut gewählt werden
- ▶ Inversed document frequenz
  - ▶ Je häufiger Wort in allen Dokumenten enthalten  $\Rightarrow$  unbedeutend
  - ▶ Wert für *max-df* muss gut gewählt werden
- ▶ Dadurch entsteht Documents  $\times$  Features Matrix
- ▶ Max. Features werden bestimmt.

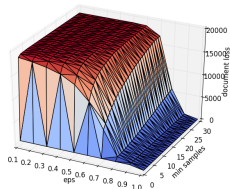
# GUTE METRIC FINDEN



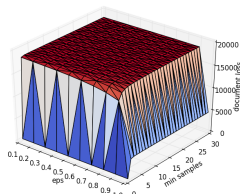
Euclidian



L2

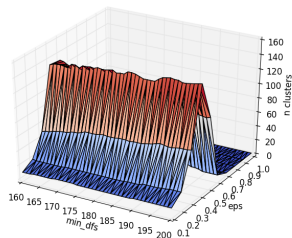
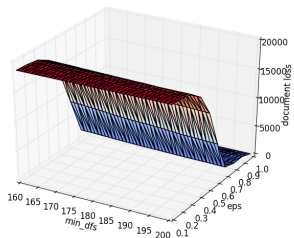
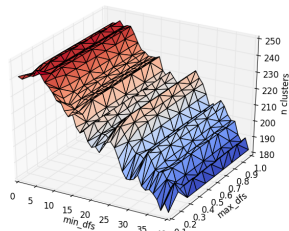
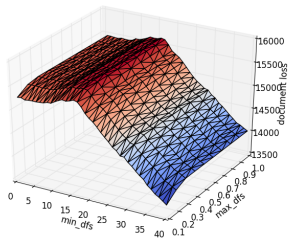


Cosine

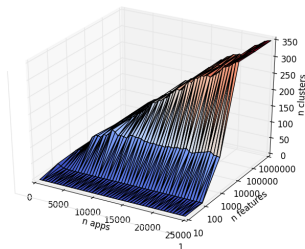
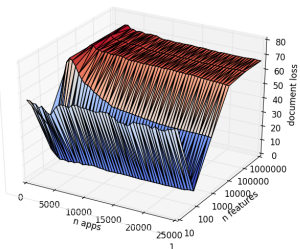
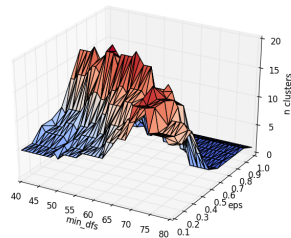
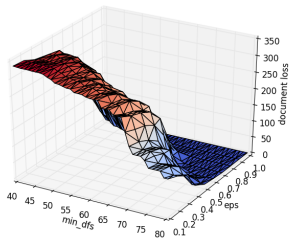


Minkowski

# GUTE PARAMETER FINDEN - TESTDATEN



## GUTE PARAMETER FINDEN - GOOGLE PLAY DATEN



## GUTE PARAMETER FINDEN

- ▶ max-df: 0.01
- ▶ min-df: 0.005
- ▶ eps: 0.45
- ▶ min-samples: 30
- ▶ features: 1500

⇒ 42 Cluster

⇒ Mehr als 50% Rauschen

⇒ 1 Cluster viel zu groß

# KOMBINATION MIT ANDEREN ALGORITHMEN

## ► K-Means

- Anzahl Cluster aus DBSCAN → **mäßiger** Erfolg
- Anzahl GP Kategorien → **mäßiger** Erfolg

## ► Classifier

- DecisionTree → **miserabler** Erfolg
- BernoulliNB → **miserabler** Erfolg
- MLP → **miserabler** Erfolg
- AdaBoost → **miserabler** Erfolg
- KNeighbors → **akzeptabler** Erfolg

⇒ Kein Verlust mehr durch Rauschen

⇒ Zu großer Cluster wurde noch größer

⇒ Cluster beinhaltet mehr als 50% apps



# HIERARCHICAL DBSCAN

Aufteilung von zu großen Cluster in kleinere.

⇒ Sprengt den Arbeitsspeicher.

*Dies liegt an der mieserablen Implementierung in SKLearn. Es ist besser, wenn du's selbst implementierst.  
- Viele Leute bei Stackoverflow*

Eigene Variante in Kombination mit KNeighbors:

- ▶ Zu große Cluster werden erneut mit DBSCAN geclustert (kleineres Epsilon)
- ▶ Dabei entstandenes Rauschen wird mithilfe KNeighbors neu verteilt

⇒ Clusterqualität wurde schlechter, kein guter Erfolg

# BEWERTUNG DER APPS

Die Apps werden nach dem Einfluss auf die Privatsphäre bewertet.

1. Sammeln der Berechtigungen innerhalb eines Clusters

## Permissions

<b>0</b>	<b>4</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
----------	----------	----------	-----------	-----------	-----------

Mit den Berechtigungen:

<b>ID</b>	<b>Name</b>
0	In-App-Purchases
4	Calender
9	Pictures/Media/Files
10	Storage
11	Camera
12	Microphone

# BEWERTUNG DER APPS

## 2. Berechnung der Gewichtung

Besteht aus zwei Teilen:

- ▶ Relative häufigkeit von App die diese Berechtigung **nicht** haben

0.4	0.8	0.6	0.2	0.0	0.6
-----	-----	-----	-----	-----	-----

- ▶ Bösheit der Berechtigung

0.1	0.6	0.1	0.1	0.9	0.9
-----	-----	-----	-----	-----	-----

# BEWERTUNG DER APPS

Diese werden miteinander multipliziert.

Permissions

<b>0</b>	<b>4</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
0.04	0.48	0.06	0.02	0.0	0.54

## 3. Füllen der Matrix

Permissions

Apps	ID	<b>0</b>	<b>4</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
	<b>14</b>	0.04	0.0	0.0	0.02	0.0	0.54
	<b>42</b>	0.0	0.48	0.06	0.0	0.0	0.0
	<b>145</b>	0.04	0.0	0.0	0.02	0.0	0.0
	<b>465</b>	0.04	0.0	0.06	0.02	0.0	0.54
	<b>1010</b>	0.0	0.0	0.0	0.02	0.0	0.0

## BEWERTUNG DER APPS

## 4. Aufsummieren der Werte

		Permissions						
Apps	ID	0	4	9	10	11	12	$\Sigma$
	14	0.04	0.0	0.0	0.02	0.0	0.54	<b>0.6</b>
	42	0.0	0.48	0.06	0.0	0.0	0.0	<b>0.54</b>
	145	0.04	0.0	0.0	0.02	0.0	0.0	<b>0.06</b>
	465	0.04	0.0	0.06	0.02	0.0	0.54	<b>0.66</b>
	1010	0.0	0.0	0.0	0.02	0.0	0.0	<b>0.02</b>

# BEWERTUNG DER APPS

## 5. Aufteilen in 3 Gruppen mithilfe K-Means

Apps

ID	$\Sigma$
14	0.6
42	0.54
145	0.06
465	0.66
1010	0.02

- ▶ Gut - Grün
  - ▶ 80 - 120 degree
- ▶ Mittel - Gelb
  - ▶ 30 - 79 degree
- ▶ Schlecht - Rot
  - ▶ 0 - 29 degree

```
# 0 - 100
value = 100 - ((app_values[i] - min_value) *
               100.0) / (max_value - min_value)
# min_range - max_range
value = (value * (color_range[1] -
                 color_range[0]) / 100) + color_range[0]
```