Master Thesis

im Rahmen des
Universitätslehrganges „Geographical Information Science & Systems“
(UNIGIS MSc) am Interfakultären Fachbereich für GeoInformatik (Z_GIS)
der Paris Lodron-Universität Salzburg

zum Thema

„Zielgruppenorientierte Werbung im öffentlichen Raum mit Hilfe räumlicher und zeitlicher Themenextraktion aus sozialen Netzwerken“

vorgelegt von

B.Eng. Verena Kettmann
ID 104210, UNIGIS MSc Jahrgang 2015

Zur Erlangung des Grades
„Master of Science (Geographical Information Science & Systems) – MSc(GIS)”

Köln, 02.01.2018

Eigenständigkeitserklärung

„Ich versichere, diese Master Thesis ohne fremde Hilfe und ohne Verwendung anderer als der angeführten Quellen angefertigt zu haben, und dass die Arbeit weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegen hat oder sonst wie veröffentlicht worden ist. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind entsprechend gekennzeichnet."

Köln, 02.01.2018 

Ort, Datum B.Eng. Verena Kettmann

Kurzfassung

Mobil-nutzergenerierte Daten aus sozialen Netzwerken bieten ein großes Potential, wenn es darum geht Zielgruppen zu definieren und zu lokalisieren. Menschen schreiben über Themen die sie gerade beschäftigen und teilen sie mit der ganzen Welt. Diese Informationen können für die Werbebranche von großer Bedeutung sein, insbesondere für die Planung von digitaler Werbung im öffentlichen Raum. Mit dieser Arbeit soll überprüft werden, ob aus der Vielzahl an Nutzerbeiträgen sozialer Netzwerke thematische Cluster extrahiert werden können, die sich sowohl räumlich als auch zeitlich klar abgrenzen lassen. Als Quelle wird auf die populärsten sozialen Netzwerke in Deutschland zurückgegriffen, die eine offene API anbieten und darüber hinaus die notwendigen Informationen für diese Arbeit liefern: Den Textbeitrag, das Datum und die Uhrzeit des Beitrags sowie die geografische Position, an welcher der Textbeitrag verfasst wurde. Dies trifft auf die Netzwerke Twitter, Instagram und Flickr zu. Die Beiträge dieser sozialen Netzwerke sollen mit Hilfe von Methoden des maschinellen Lernens geclustert werden um Interessen-Themen zu extrahieren. Im Fokus steht hier die Methode Latent Dirichlet Allocation, welche auf Wahrscheinlichkeiten basiert. Kombiniert wird die Methode mit dem Gibbs-Sampling-Algorithmus, welches das Wahrscheinlichkeitsmodell iterativ annähert. Die Arbeit beschreibt, wie die Daten der sozialen Netzwerke gesammelt und gespeichert werden. Die gesammelten Textbeiträge werden, bevor sie mittels der Methode Latent Dirichlet Allocation analysiert werden können, bereinigt und aufbereitet. Für diesen Zweck kommen mehrere Methoden zum Einsatz wie z.B. Rechtschreibkorrektur oder Lemmatisierung. Da die Textbeiträge der sozialen Netzwerke in 5 ausgewählten Großstädten Deutschlands gesammelt werden, werden diese Methoden mit deutschsprachigen Bibliotheken und Wörterbüchern durchgeführt, als auch, bedingt durch den hohen Tourismus-Anteil in den Großstädte, mit englischsprachigen Bibliotheken und Wörterbüchern. Die resultierenden Ergebnisse werden mit Hilfe der räumliche Autokorrelation Local Moran I kartographisch ausgewertet und interpretiert. Die Arbeit zeigt, dass eine Auswertung mit den hier beschriebenen Methoden auf Zeitschienen-Ebene im deutschsprachigen Raum möglich ist, jedoch mit erhebliche Qualitätseinbußungen gerechnet werden muss, da die Anzahl der gesammelten Textbeiträge, bedingt durch das hohe Datenschutzbewusstsein in Deutschland, verhältnismäßig gering sind.

Abstract

Mobile user-generated content provides a huge potential for definition and localization of target-groups. People are writing about topics, which they are interested in and share them with the whole world. These information can have a highly important impact for the advertising industry, especially for the planning of digital advertisement in public space. The purpose of this work is to evaluate, if it is possible to extract thematic clusters from the large amount of social network postings and if it is possible to clearly differentiate them spatially and temporally. The sources will be the most popular social networks in Germany that offer an open API which, moreover, deliver the necessary information for this work: The text contribution, the date and time as well as the geographic position of the location where the text contribution was send. The networks Twitter, Instagram and Flickr meet the requirements. The text contributions of these social networks will be clustered with methods of machine learning to extract topics of interest. The focus will be laid on the method Latent Dirichlet Allocation which is based on probabilities. The method will be combined with the Gibbs-Sampling algorithm, which approximates the probability model. It will be described, how the social media data will be collected and saved. The saved text contributions will be prepared and corrected before the analyzation with Latent Dirichlet Allocation starts. For this purpose several methods will be used e.g. spelling correction or lemmatization. The text contributions will be collected in 5 selected large cities in Germany. Therefore this methods will be executed with German libraries and dictionaries. Regarding to the high tourism ratio, English libraries and dictionaries will be also used. The results will be cartographically analyzed and interpreted with the help of the spatial autocorrelation Local Moran I. This work shows, that an evaluation with the presented methods in the German speaking area for specific timeslots is possible. Regarding to the small number of text contributions, which is caused by the high data protection awareness in Germany, it has to be anticipated with losings of quality.

Inhaltsverzeichnis

1. Einleitung 1

1.1 Problemstellung 1

1.2 Ziel der Arbeit 2

1.3 Methodik 3

1.4 Verwandte Arbeiten 4

1.5 Aufbau der Arbeit 6

2. Grundlagen 8

2.1 Topic-Models 8

2.2 Latent Dirichlet Allocation 8

2.3 Gibbs-Sampling 11

2.4 Tokenisierung 12

2.5 Bag-of-Words 13

2.6 Stemming und Lemmatisierung 13

3. Datengrundlage 15

3.1 Eingrenzung der Zieldaten 15

3.2 Problematik der Datengewinnung 16

3.3 Stammdaten digitaler Werbeträger 18

3.4 Data-Crawler 18

4. Systemarchitektur 20

4.1 Übersicht 20

4.2 Datenbank 20

4.3 Programmiersprachen und Bibliotheken 21

5. Datenaufbereitung 23

5.1 Grundlegende Aufbereitungen 23

5.2 Blacklist 23

5.3 Bot-Beiträge 24

5.4 Spracherkennung 25

5.5 Rechtschreibkorrektur 26

5.6 Stoppworte 27

5.7 Lemmatisierung 27

5.8 Tokenization und Bag-of-Word 28

5.9 Unicode-Emoticons 29

5.10 Implementierung 30

6. Themenextraktion 33

6.1 LDA-Model 33

6.2 Themenspeicherung 35

7. Evaluierung 37

7.1 Aufbau der Testdaten 37

7.1.1 Eingrenzung Flickr/Twitter 37

7.1.2 Übersicht Testdaten 38

7.1.3 Blacklist-Parameter 39

7.2 Zielgruppen-Themen 39

7.3 Räumliche Auswertung Themen – Textbeiträge am Beispiel „Fashion/Mode“ 42

7.3.1 Interpolation 42

7.3.2 Standortempfehlung mit local Moran I 53

7.4 Diskussion und Bewertung 68

Quellenverzeichnis 69

8. Anhang I

Anhang 1: Quellcode I

Anhang 2: Extrahierte Themencluster XVII

1. Einleitung

1.1 Problemstellung

Wer ein Produkt bewerben möchte, sollte sich zunächst über ein paar wichtige Fragen klar werden. Was möchte ich wo und wie bewerben und vor allem, wen möchte ich mit meiner Werbung erreichen? Eine wichtige Entscheidung ist in diesem Zusammenhang auch die Wahl des Werbemediums. Mit einer Anzeige in einer Tageszeitung kann eine ganz andere Werbewirkung erzielt werden, als wenn das gleiche Motiv auf einer großen Plakatwand zu sehen ist. Letzteres fällt in den Bereich der Außenwerbung. Außenwerbung ist ein Zweig der Werbebranche, der primär auf Werbung im öffentlichen Raum spezialisiert ist. Dies umfasst große klassische Plakatflächen die an Straßen stehen aber auch experimentelle Formen der Werbung wie z.B. Werbeaufdrucke auf Straßenbahnen. Die räumliche Platzierung der Werbung ist in der Regel nicht willkürlich, sondern das Resultat von vorangegangenen Analysen. Je nach Kundenwunsch und Werbemotiv ist es Ziel, das bestmögliche Potential aus dem vorgegebenen Werbeetat zu gewinnen, indem die gewünschte Zielgruppe direkt dort angesprochen wird, wo sie sich mit einer hohen Wahrscheinlichkeit aufhält. Oft wird in diesem Zusammenhang der Begriff Geomarketing verwendet. Für ein erfolgreiches Geomarketing sind sowohl räumliche als auch statistische Daten unerlässlich. Nur wenn es Daten darüber gibt, wo sich eine bestimmte Zielgruppe aufhält, kann Diese mit möglichst geringen Streuverlusten angesprochen werden. Mit dem technischen Fortschritt wandelt sich das Bild der Außenwerbung zunehmend. Analoge Werbeflächen, welche manuell beklebt und in einem Zyklus von durchschnittlich 10,5 Tagen durch ein neues Motiv ersetzt werden, werden mehr und mehr durch digitale Screens ausgetauscht. Ganze Werbenetze entstehen an neuen und vorher undenkbaren Orten und ermöglichen so eine ganz neue Form der Außenwerbung. Werbung auf digitalen Screens kann viel gezielter und flexibler eingesetzt werden, als es in der klassischen Außenwerbung je möglich war. Es können nicht nur viel mehr Werbemotive auf einem einzigen Screen gezeigt werden, sondern der Kunde kann auch ein Zeitfenster bestimmen, in welchem er sein Werbemotiv ausstrahlen möchte. Sowohl über den Tag verteilt, also auch über die Woche. Diese Form der Buchungseinheit

wird Zeitschiene genannt. Die meisten Anbieter von digitalen Werbeträgern definieren eine Zeitschiene als einen 3-stündigen Slot, in welchem eine Programmschleife die eingebuchten Werbemotive wiederholt. Diese neue Möglichkeit, Zielgruppen ganz gezielt ansprechen zu können bringt auch neue Probleme mit sich. Die Frage lautet nun nicht mehr nur „wo befindet sich meine Zielgruppe“, sondern auch „wann hält sie sich dort auf“. Es gibt eine Vielzahl an Daten des statistischen Bundesamtes und Befragungsstudien die in den Raum kumuliert werden können umso die Frage des „wo“ gut beantworten zu können. Jedoch lassen alle diese Datensätze keine zeitliche Betrachtung zu. Um für die Planung von digitalen Werbeträgern eine Zielgruppe sowohl räumlich, als auch zeitlich bestimmen zu können, muss ein neuer Ansatz entwickelt werden, dessen Datengrundlage es erlaubt, Rückschlüsse über diese Informationen gewinnen zu können. Mobil-nutzergenerierte Daten aus sozialen Netzwerken stellen eine mögliche Quelle für diese Informationen dar.

1.2 Ziel der Arbeit

Mobil-nutzergenerierte Daten bieten eine ideale Datengrundlage zur räumlichen und zeitlichen Zielgruppenbestimmung. Sie sind direkt von der potentiellen Zielgruppe verfasste Textbeiträge, die neben den eigentlichen thematischen Inhalten auch die Uhrzeit sowie die geografische Position liefern von welcher die Nachricht gesendet wurde. Dies setzt natürlich voraus, dass der Anwender seine Erlaubnis dazu erteilt hat, seine geographische Position zu veröffentlichen. Ziel dieser Arbeit ist es zu überprüfen, ob diese Daten verwendet werden können, um mit Hilfe von Methoden des maschinellen Lernens thematische zu bilden, die sowohl räumlich als auch zeitlich differenzierbar sind. Es soll überprüft werden, ob die gewählten Methoden auf den deutschsprachigen Raum angewendet werden können. Da es die Motivation dieser Arbeit ist, die Planbarkeit von Werbekampagnen für digitalen Werbeträgern zu verbessern, werden die thematischen Cluster nicht flächendeckend in Deutschland gewonnen, sondern konzentrieren sich räumlich um die Standorte von digitalen Werbeträgern. Dies sind in der Regel die Großstädte Deutschlands. In dieser Arbeit wird das Verfahren exemplarisch auf Basis von Daten der Großstadt Berlin durchgeführt, um die Machbarkeit zu überprüfen. Die Wahl der Stadt wird in Kapitel *7.1 Aufbau der Testdaten* begründet.

1.3 Methodik

Für die Bildung der thematischen Cluster werden Methoden des maschinellen Lernens verwendet. Diese sind für diesen Anwendungszweck besonders geeignet, da eine sehr große Anzahl an Texten objektiv analysiert werden soll. Die Masse der Daten würde eine manuelle Auswertung nicht erlauben. Die Methode die hier eingesetzt wird, ist „Latent Dirichlet Allocation“ (LDA). Dies ist ein Verfahren, welches auf der Berechnung von Wahrscheinlichkeiten basiert. Bei LDA wird für jedes Wort in einem Dokument eine Wahrscheinlichkeit berechnet die indiziert, wie themenbildend das Wort für das Dokument ist. Ein Dokument kann aus mehreren Themen bestehen und diese Themen können in mehreren Dokumenten vorkommen. Vor diesem Hintergrund können dokumentübergreifend thematische Cluster gebildet werden *(Blei et al., 2003*). Textbeiträge sozialer Netzwerke, welche in Deutschland verfasst werden, beschränken sich nicht ausschließlich auf Text der deutschen Sprache, sondern sind größtenteils eine Mischung aus deutschen und englischen Texten. Bedingt durch den hohen Tourismus in deutschen Großstädten ist ein Teil der Textbeiträge auch in anderen Sprachen verfasst, aber der größte Teil beschränkt sich auf diese beiden Sprachen. Dies bedeutet, dass die gewählte Methode mit mehr als nur einer Sprache kompatibel sein muss. LDA ist für diese Problematik besonders gut geeignet, da die Methode auf einem rein mathematischer Ansatz basiert und somit sowohl sprachunabhängig, als auch sprachenübergreifend eingesetzt werden kann (*Mimno et al., 2009*). Somit können Texte beliebiger Sprache verarbeitet werden, die innerhalb eines zu analysierenden Texte-Korpus auch unterschiedliche Sprachen aufweisen können. Mit LDA werden einmalig die wahrscheinlichkeitsbasierenden thematischen Cluster ermittelt. Um hier aber ein optimales Ergebnis zu erreichen, wird LDA in der Regel in Kombination mit einer iterativen Methode angewendet, welche die thematischen Cluster sukzessive annähert und optimiert. Oft wird in diesem Fall der Algorithmus „Gibbs-Sampling“ eingesetzt, der nach Eingabe einiger Startparameter, die Wortverteilungen den Themen annähert (*Griffiths and Steyvers, 2004*). Da das Gibbs-Sampling bereits in anderen wissenschaftlichen Arbeiten gute Ergebnisse erzielt hat, wird in dieser Arbeit die Methode LDA mit Gibbs-Sampling kombiniert (vgl. *1.4 Verwandte Arbeiten*).

Bevor LDA und das Gibbs-Sampling angewandt werden, werden die eingehenden Texte zur maschinellen Auswertung aufbereitet. Dies beinhaltet die Entfernung von Stoppworten, Rechtschreibkorrekturen, Lemmatisierung, Tokenization und Bag-of-Word-Matrizen. Durch diese Aufbereitungen werden die Texte so weit wie möglich homogenisiert um ein optimaleres Ergebnis zu erzielen. Die hier genannten Methoden werden in dem Kapitel *2. Grundlagen* näher erläutert.

1.4 Verwandte Arbeiten

Nutzergenerierte-Daten von sozialen Netzwerken, insbesondere Twitter-Daten, sind Gegenstand vieler wissenschaftlicher Arbeiten. Die Daten sind in der Regel frei zugänglich und in einer großen Anzahl, von vielen unterschiedlichen Menschen bzw. Personengruppen, verfasst. Dies macht sie besonders interessant für wissenschaftliche Analysen und Auswertungen unterschiedlicher Themengebieten. Die meisten Arbeiten befassen sich damit, die verfassten Textnachrichten auszuwerten und nutzen dabei unterschiedlichste Methoden. Cheng, Caverlee und Lee haben 2010 eine Arbeit veröffentlicht, welche sich darauf konzentriert, Textbeiträge der Plattform Twitter zu analysieren, um Rückschlüsse über die geographische Position des Verfassers der Nachricht ziehen zu können. Dabei werden mehrere Methoden angewendet, die alle gemein haben, dass der Inhalt der Textbeiträge geclustert wird. Anschließend werden die Ergebnisse mit weiteren Informationen wie z.B. der IP-Adresse oder Informationen externer Wissensdatenbanken angereichert, um die Lokalisation zu verfeinern bzw. zu optimieren. Die Methoden, die Cheng, Caverlee und Lee verwenden, sind jedoch nicht aus dem Bereich der Topic-Modeling-Algorithmen (Begriff wird in Kapitel *2.1 Topic-Models* näher beschrieben). Vielmehr findet ein Clustering über die einzelnen Worte der Textbeiträge statt, welche kategorisiert und in Kontext gebracht werden. Dazu werden im Vorfeld Worte definiert, welche Rückschlüsse über ein Gebiet geben können. Als Beispiel wird das Wort „howdy“ genannt, welches eine umgangssprachliche Begrüßung in Texas ist. Textbeiträge, die dieses Wort beinhalten legen den Schluss nahe, dass der Verfasser sich zum Zeitpunkt der Erstellung wahrscheinlich in Texas befunden hat (*Cheng et al., 2010*). Die Methode, welche am häufigsten in wissenschaftlichen Arbeiten angewendet wird, wenn es das Ziel ist, Textbeiträge von

Plattformen wie Twitter auszuwerten und zu clustern, ist LDA in Verbindung mit einem Annäherungs-Algorithmus wie dem Gibbs-Sampling. Je nach Motivation der jeweiligen Arbeit, werden die durch LDA gewonnen Ergebnisse mit weiteren Verfahren und Methoden aufbereitet und konkretisiert. Obwohl in vielen Arbeiten diese Methode angewendet wird, sind die Ergebnisse, je nach Aufbereitung, Zeitstand, Kontext und weiteren Analysen, immer Andere und somit attraktiv für wissenschaftliche Arbeiten. Als Beispiel kann hier die 2011 veröffentlichte Arbeit von Zhao, Jiang, Weng, He und Lim genannt werden. Hier wird der Inhalt von Zeitungsartikeln der New York Times mit Textbeiträgen der Plattform Twitter verglichen. Ziel war es herauszufinden, ob beide Quellen ähnliche Informationen liefern und somit Twitter als Nachrichtenmedium genutzt werden kann. Dazu wurden Methoden des Topic-Modeling wie LDA verwendet um die Inhalte der Zeitungsartikel, als auch der Twitter-Textbeiträge, zu clustern bzw. zu extrahieren. Kombiniert wird die Methode mit Gibbs-Sampling um das Wahrscheinlichkeitsmodell anzunähern. Für den eigentlichen Vergleich der Ergebnisse werden im Anschluss Methoden des Textmining herangezogen (*Zhao et al., 2011*). Die Arbeit zeigt, dass sich die gewählten wissenschaftlichen Methoden sehr gut eignen um große Textsammlungen automatisiert zu verarbeiten und zu analysieren. Die bisher betrachteten Arbeiten haben ausschließlich Textbeiträge analysiert, welche in englischer Sprache verfasst sind. Besonders LDA ist allerdings auch dazu geeignet, Texte sprachunabhängig zu verarbeiten. Dies zeigt die Arbeit von Yang, Xu und Li aus dem Jahr 2011. Die Arbeit untersucht 3 unterschiedliche Methoden, mit welchen die Interessen von Benutzern identifiziert werden sollen, indem die von ihnen verfassten Textbeiträge aus sozialen Netzwerken geclustert werden. Das soziale Netzwerk, welches hier näher betrachtet wird, ist nicht wie bei so vielen anderen Arbeiten die Plattform Twitter, sondern Weibo. Dies ist ein sehr populäres soziales Netzwerk aus China, in welchem die Texte überwiegend in chinesischer Sprache verfasst werden. Eines der 3 Verfahren, welche die Interessen der Benutzer identifizieren sollen, ist LDA. Die Arbeit von Yang, Xu und Li zeigt, dass es mit Hilfe der Methode möglich ist, die Interessen-Themen der Benutzer einzugrenzen, auch wenn sie in einer Sprache wie Chinesisch geschrieben sind (*Zheng Yang et al., 2011*).

1.5 Aufbau der Arbeit

Diese Arbeit ist in 7 Kapitel unterteilt. Das erste Kapitel stellt die Einleitung dar. In diesem Kapitel wird aufgezeigt, welche Motivation hinter dieser Arbeit steht. Es wird die Problemstellung skizziert und die Ziele der Arbeit näher beleuchtet. Neben den Methoden, die zur Problemlösung führen sollen, wird ein Überblick über verwandte Arbeiten gegeben. Der Einleitung folgt in Kapitel 2 eine nähere Betrachtung der theoretischen Grundlagen, die den anzuwendenden Methoden zugrunde liegen. Es werden wichtige Begriffe erklärt und die Funktionsweise von LDA sowie die in Kombination angewendeten Methoden wie z.B. Gibbs-Sampling oder Lemmatisierung erläutert. Anschließend wird in Kapitel 3 näher auf die Daten eingegangen, die den hier durchgeführten Analysen zugrunde liegen. Es wird beschrieben, welche mobil-nutzergenerierten Daten verwendet werden. Ebenso wird aufgezeigt, wie diese Daten gewonnen werden und welche Probleme damit einhergehen. Nachdem die Datengrundlage geklärt ist, folgt in Kapitel 4 eine Beschreibung der Systemarchitektur. Es wird skizziert, welche Komponenten verwendet werden, um die Analyse mit LDA durchzuführen. Dies umfasst sowohl eine Übersicht der verwendeten Datenbank und Programmiersprache, als auch alle Packeges bzw. Bibliotheken, die von Drittanbietern eingebunden werden. Anschließend wird in Kapitel 5 näher beschrieben, welche Schritte unternommen werden, um die gewonnenen Daten, vor der Analyse mit LDA, aufzubereiten. Es werden alle Schritte näher beschrieben und deren Notwendigkeit dargelegt. Das Kapitel schließt mit einer Gesamtübersicht aller durchgeführten Prozesse und beschreibt die eigentliche Implementierung. Dem folgt in Kapitel 6 eine Beschreibung, wie die Themen mit Hilfe von LDA und den entsprechenden LDA-Modellen gewonnen werden. Hier wird ebenfalls kurz auf die Implementierung eingegangen und der Zusammenhang zwischen dem LDA-Model und den gespeicherten Ergebnissen beschrieben. In Kapitel 7 werden die zuvor beschrieben Methoden und Techniken an einem praktischen Beispiel angewendet um eine nachfolgende Evaluierung zu ermöglichen. Diese Evaluierung wird exemplarisch für ein zuvor definiertes Gebiet durchgeführt. Es wird der Testaufbau beschrieben und wie die Durchführung aussieht. Hier wird neben der Themengewinnung ebenfalls näher auf die zentrale Kernfrage dieser Arbeit eingegangen. Es soll überprüft werden, ob mit Hilfe von Daten aus sozialen Netzwerken und maschineller

Themenextraktion Zielgruppen-informationen gewonnen werden können, die dabei helfen digitale Außenwerbung zu planen. Um dies zu beantworten wird nach der Themenextraktion eine räumliche Auswertung der Ergebnisse durchgeführt und die dort gewonnenen Ergebnisse im Anschluss diskutiert.

2. Grundlagen

2.1 Topic-Models

Topic-Models ist ein Oberbegriff für eine Gruppe von Algorithmen die es erlauben, mit Hilfe von statistischen Analysen, Rückschlüsse über die thematische Struktur einer Ansammlung von unstrukturierten Texten zu ziehen. Neben den thematischen Informationen decken diese Algorithmen auf, wie die Texte und Themen zusammenhängen und wie sie sich im Laufe der Zeit verändert haben. Alle Algorithmen die der Gruppe der Topic-Models zugeordnet werden können haben gemeinsam, dass sie auf eine große Anzahl von Texten angewendet werden können (*Blei, 2012*).

2.2 Latent Dirichlet Allocation

Eine weit verbreitete Methode der Topic-Models ist Latent Dirichlet Allocation (LDA). Die folgenden Ausführungen dieses Kapitels beziehen sich auf die in 2012 von Blei geschilderten Ergebnisse zu diesem Thema (*Blei, 2012*). LDA ist ein probabilistisches Verfahren, welches auf dem Kerngedanken basiert, dass es beim Topic-Modeling zwei Arten von Variablen gibt: Die beobachtbaren und die nicht-beobachtbaren (latenten) Variablen. Als beobachtbar werden Dokumente und deren Worte eingeordnet. Nicht-beobachtbar sind die eigentlichen Themen, die mit Hilfe von LDA gefunden werden sollen. Um nachvollziehen zu können, wie diese Themen mit Hilfe von LDA entstehen, ist es notwendig die Beziehung zwischen diesen Variablen zu verdeutlichen:

- Ein *Dokument* besteht aus mehreren Themen. Dabei fallen den Themen unterschiedliche Gewichtungen zu, welche aussagen wie themenbildend sie für das Dokument sind.
- Ein *Thema* stellt eine Wahrscheinlichkeitsverteilung über das gesamte Vokabular aller Dokumente dar. Dabei kann die Wahrscheinlichkeit niemals den Wert 0 haben. Für jedes Wort wird eine Wahrscheinlichkeit ermittelt, mit der es einem Thema angehört - sei sie noch so klein. Ein Beispiel: Die Worte „Banane“ und „Schinken“ gehören mit einer hohen

Wahrscheinlichkeit dem Thema „Nahrung/Essen“ an. Wo hingegen die Wahrscheinlichkeit gering ist, dass das Wort „Eifelturm“ auch in dieses Thema fällt.

- Ein Wort kann mehreren Themen angehören. Dabei kann das gleiche Wort mit ähnlich hohen Wahrscheinlichkeiten mehreren Themen zugeordnet werden. Das Wort „Banane“ hat zwar eine hohe Wahrscheinlichkeit dem Thema „Nahrung/Essen“ anzugehören, aber gehört ebenso mit hoher Wahrscheinlichkeit dem Thema „Obst“ an.

Abbildung 1 zeigt die hier beschriebenen Zusammenhänge des LDA-Models, wie sie Blei 2003 definiert hat.

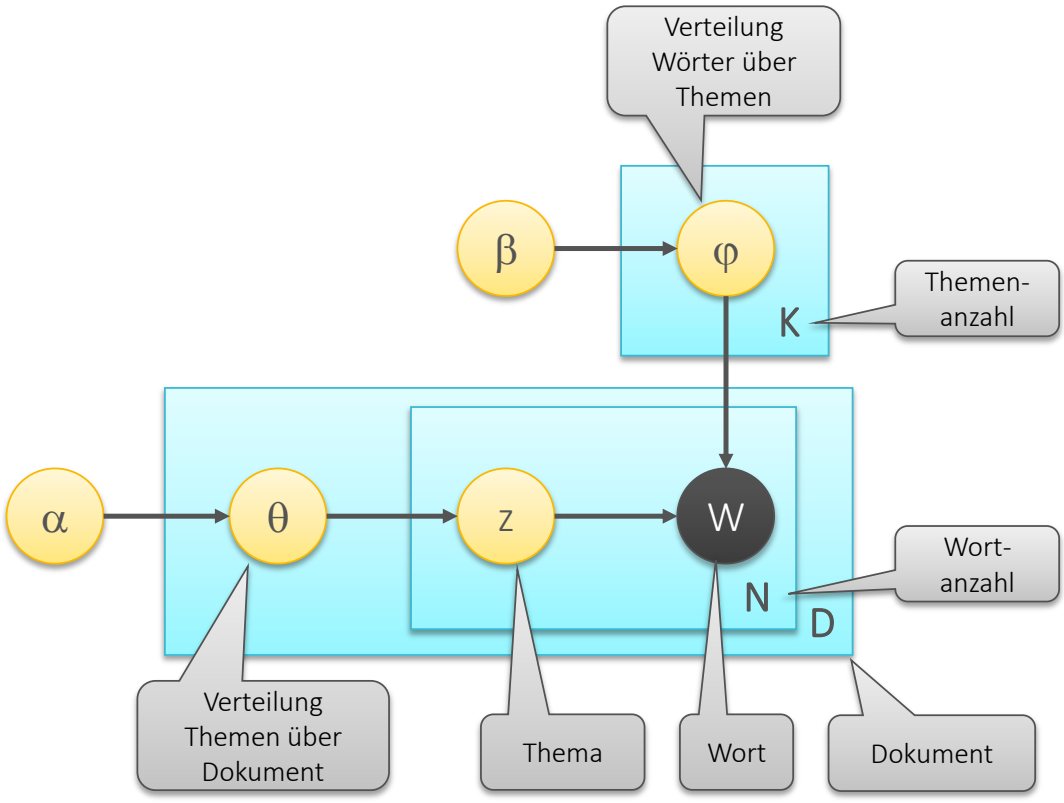


Abbildung 1: Grafisches Darstellung des LDA-Models (Blei et al., 2003)

LDA arbeitet mit Bag-of-Word (siehe Kapitel *2.5 Bag-of-Words)* Matrizen. Demnach werden in dem Verfahren keine semantischen Rückschlüsse aus den Texten gewonnen. Vielmehr wird davon ausgegangen, dass die Reihenfolge der Worte in einem Dokument nicht wichtig ist. Ein Dokument wird nicht als literarisches Werk mit vielschichtigem Kontext und

Bedeutung angesehen. Wahrscheinlichkeitsbasierte Topic-Models wie LDA arbeiten auf der Annahme, dass ein zufälliger zweistufiger Prozess zu der Entstehung der Dokumente geführt hat. In diesem Prozess wird zunächst zufällig generiert, aus welchen Themen sich jedes Dokument zusammensetzt. Danach wird jedes Wort eines Dokumentes zufällig den darin enthaltenen Themen zugordnet. Bei diesem Prozess sind die Themen und ihre Verteilung der Ausgangspunkt.

Unter dieser Annahme zur Entstehung von Dokumenten wird der Prozess umgedreht, um aus einer gegebenen Sammlung von Dokumenten die Themen und ihre Verteilung zu ermitteln. Das Problem bei diesem Ansatz besteht darin, dass die Anzahl aller möglichen Themenverteilungen sehr groß ist. Alle möglichen Themenverteilungen können daher nicht berechnet werden. Vielmehr nähert LDA in Verbindung mit dem Gibbs-Sampling die Ergebnisse immer weiter an, bis ein zufriedenstellendes Ergebnis erreicht ist. Eine Annäherung ist jedoch nur möglich, wenn die Anzahl der Themen die ermittelt werden sollen, im Vorfeld festgelegt wurde.

Abbildung 2 verdeutlich die hier beschriebenen Prozesse und deren zu erwartende Ergebnisse.

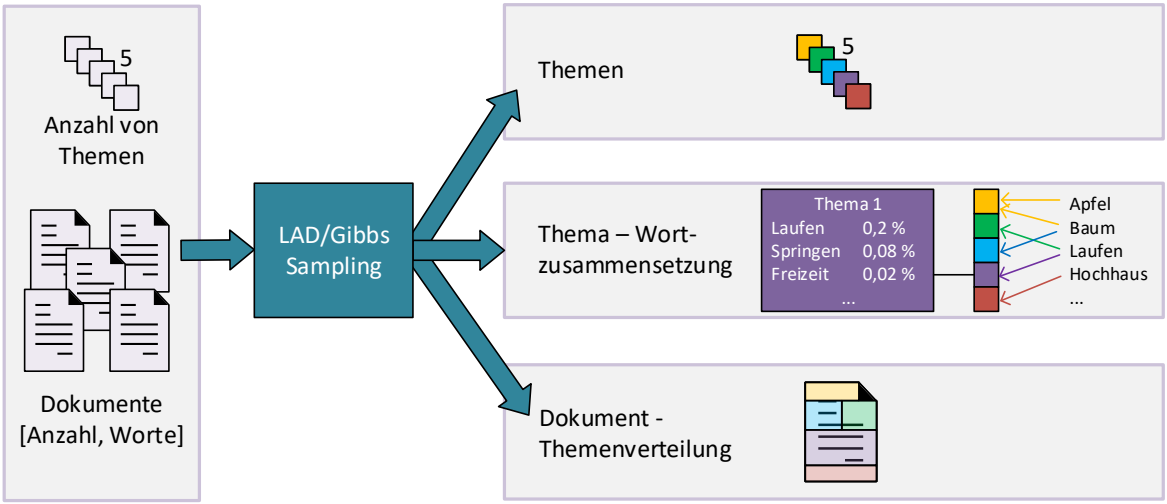


Abbildung 2: LDA/Gibbs Sampling - Eingabe/Ergebnisse

2.3 Gibbs-Sampling

Das Gibbs-Sampling ist ein Algorithmus, welcher mit der Markov-Chain-Monte-Carlo-Methode (MCMC-Methode) arbeitet. Mit Hilfe des Gibbs-Sampling können Stichproben aus mehrdimensionalen Verteilungen generiert werden, bei welchen nur einige der bedingten Verteilungen bekannt sind (*Casella and George, 1992*). Es ist ein iteratives Verfahren, bei dem aus einer großen Sammlung an Dokumenten Themen extrahiert werden können. Dazu wird im ersten Schritt die Anzahl der Themen festgelegt, welche durch das Gibbs-Sampling in einer gegebenen Dokumentsammlung ermittelt werden sollen. Anschließend wird jedem Wort aller Dokumente ein beliebiges Start-Thema zugewiesen. Nach dieser Initialisierung beginnt der eigentliche iterative Prozess. In jedem Iterationsschritt wird die Wahrscheinlichkeit berechnet, mit der ein Dokument einem Thema zugewiesen werden kann, sowie die Wahrscheinlichkeit mit der ein Wort einem Thema zugeordnet ist. Dies geschieht unter der Annahme, dass Wörter, die zusammen in einem Dokument vorkommen, mit einer höheren Wahrscheinlichkeit auch dem gleichen Thema angehören. Nach jeder Berechnung der Wahrscheinlichkeiten werden die Werte des vorherigen Iterationsschritts überschrieben. Dies wird im Idealfall solange wiederholt, bis sich die Wahrscheinlichkeiten Werten annähern, die sich bei jeder Iteration nur noch leicht verändern. Die Ergebnisse sind demnach stark von der Anzahl der Iterationen abhängig, sowie der Anzahl der zu ermittelnden Themen (*Blei, 2012*). Abbildung 3 zeigt den hier beschriebenen Prozess des Gibbs-Samplings in einer vereinfachten Darstellung.

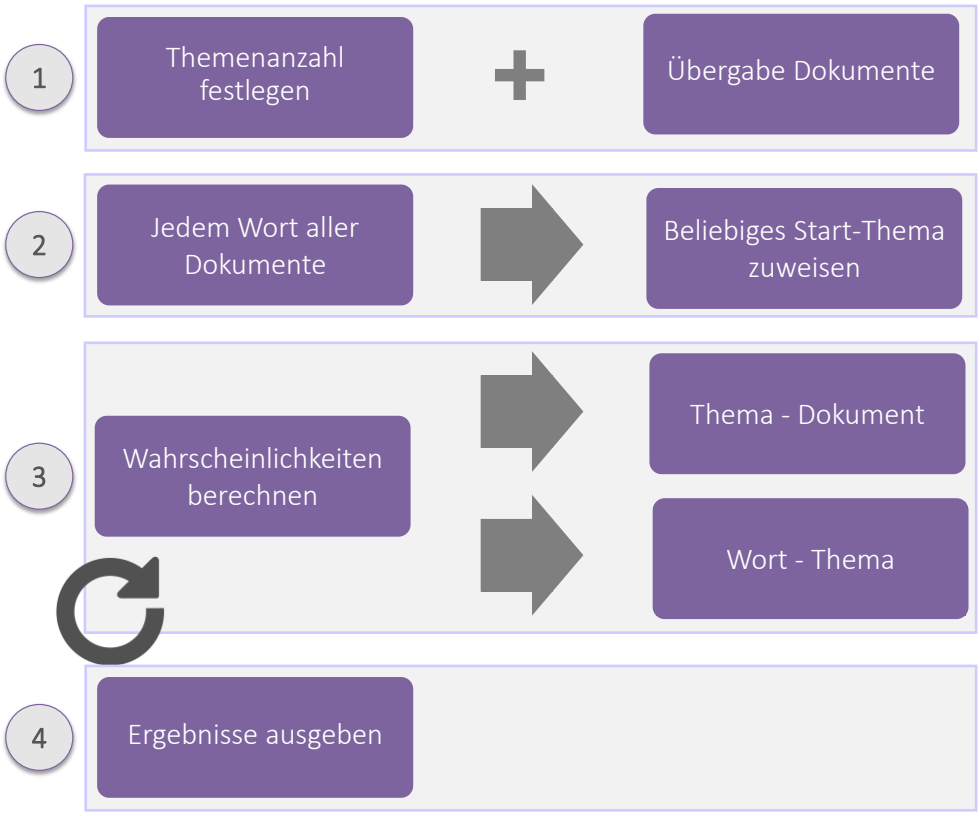


Abbildung 3: Schematische Darstellung - Gibbs-Sampling

2.4 Tokenisierung

Als Tokenisierung bezeichnet man den Prozess, wenn ein eingegebener Text in seine einzelnen Elemente zerteilt wird. In den meisten Fällen handelt es sich bei den Elementen um Satzzeichen oder Wörter. Handelt es sich bei den Elementen um Zeichenketten, nennt man diese auch Token (*Grefenstette and Tapanainen, 1997*). Neben den Elementen ermittelt der Prozess auch die Wortart (part-of-speech, POS), welche durch das POS-Tagging ermittelt wird. Dies bezeichnet die Bestimmung und Zuweisung der Wortart zu einem Token (*Abney, 1997*). Tokenisierung ist eines von mehreren Verfahren die eingesetzt werden um die Texte, die durch LDA und Gibbs-Sampling analysiert werden sollen, aufzubereiten.

2.5 Bag-of-Words

Ein Bag-of-Words Modell bezeichnet eine Darstellungsform, welche die Häufigkeit eines Wortes innerhalb eines Textes abbildet. Es wurde von H. Wallach entwickelt und ist oft Ausgangspunkt von Topic-Modeling-Algorithmen. Ein Bag-of-Words-Modell entspricht einer Matrix, in welcher jede Zeile ein Wort repräsentiert und jede Spalte für ein Dokument steht. In den Zellen dieser Matrix wird hinterlegt, wie häufig ein Wort in einem Dokument auftritt. Dabei wird weder die Syntax des Satzes, noch die Reihenfolge der Wörter beachtet. Die Darstellung konzentriert sich ausschließlich auf die Worthäufigkeit (*Wallach, 2006*). Tabelle 1 zeigt ein einfaches Beispiel einer Bag-of-Word-Matrix.

Tabelle 1: Beispiel Bag-of-Words Matrix

	Dokument 1	Dokument 2	Dokument 3
Apfel	2	0	9
Tiger	1	8	2
gehen	0	4	12
schlafen	11	3	1
grün	7	8	0

2.6 Stemming und Lemmatisierung

Stemming hat zum Ziel, Wörter auf ihren Wortstamm zurückzuführen. Der Wortstamm, welcher bei diesem Verfahren entsteht, muss kein tatsächliches Wort sein, dass in der jeweiligen Sprache vorkommt. Wie das resultierende Wort am Ende von einem Stemming-Prozess aussieht ist dabei abhängig vom verwendeten Algorithmus. Alle Stemming-Algorithmen haben jedoch gemeinsam, dass in verschiedenen Durchgängen Suffixe eines Wortes entfernt werden, sodass die Wörter verkürzt werden. Dies soll Cluster-Algorithmen wie z.B. LDA ermöglichen, bessere Zusammenhänge zwischen Wörtern herstellen zu können, die im Grunde auf demselben Wort basieren (*Lewandowski, 2005*).

Im Gegensatz zum Stemming wird bei der Lemmatisierung das Wort nicht verkürzt, sondern auf seine Grundform zurückgeführt. Das Ergebnis einer Lemmatisierung ist demnach ein

tatsächliches Wort, das in der jeweiligen Sprache vorkommt und in einem Duden zu finden ist. Diese Grundform nennt man Lemma oder auch Lexem. Da dieses Verfahren mit Wörterbüchern arbeitet und jedes Wort mit Diesen abgeglichen werden muss, ist es deutlich zeitaufwändiger als Stemming (*Lewandowski, 2005*).

3. Datengrundlage

3.1 Eingrenzung der Zieldaten

Ein wichtiges Kernelement dieser Arbeit ist es, Daten aus sozialen Netzwerken zu gewinnen, die als Datengrundlage zur Bildung von thematischen Cluster dienen. Dabei müssen diese Daten einige Voraussetzungen erfüllen, damit eine verwertbare Datenmenge gespeichert werden kann. Jedes soziale Netzwerk muss eine offene API anbieten die es erlaubt Textbeiträge mittels einer automatisierten Schnittstelle abfragen und dauerhaft speichern zu können. Dies trifft mittlerweile auf nahezu alle bekannten sozialen Netzwerke zu. Jedoch gibt es Unterschiede in der Form, in der Daten abgefragt werden können. Die API von Facebook z.B. liefert die Pinnwandbeiträge eines Benutzers, wenn diese explizit über die eindeutige User-ID angefragt werden und auch nur dann, wenn der Benutzer seine Erlaubnis erteilt hat. In dieser Arbeit kann diese API daher nicht genutzt werden, weil hier keine Informationen der User-IDs vorliegen (*Graph API /user/feed - Dokumentation - Facebook for Developers, 2017*). Darüber hinaus müssen die Textbeiträge neben der eigentlichen Nachricht noch weitere Informationen liefern. Um thematische Cluster innerhalb eines Zeitschienenfensters von 3 Stunden (vgl. *1.1 Problemstellung*) bilden zu können ist es notwendig, dass die API die Uhrzeit liefert, an welcher der Textbeitrag verfasst wurde. Um eine räumliche Betrachtung der thematisch Cluster zu ermöglichen, muss die API ebenso die geographische Position liefern, an der der Beitrag erstellt worden ist. Nicht jeder Textbeitrag liefert auch Koordinaten, da dies von jedem Benutzer individuell entschieden werden kann. Es werden in dieser Arbeit jedoch nur Beiträge gespeichert, dessen Koordinaten verfügbar sind und die somit geographisch verortet werden können.

Es hat sich herausgestellt, dass die API von Twitter, Instagram und Flicker die Daten-Anforderungen erfüllen. Jedoch kann Instagram trotz der guten Datengrundlage nicht verwendet werden, da Instagram zu Beginn dieser Arbeit seine „Platform Policy“ geändert hat. Die neue „Platform Policy“ unterbindet es nun Textbeiträge von Benutzern zu sammeln und dauerhaft zu speichern, wenn Instagram nicht seine explizite Genehmigung dazu erteilt

(*Platform Policy - Instagram, 2017*). Es werden daher die Textbeiträge der sozialen Netzwerke von Twitter und Flicker zur Auswertung gesammelt und gespeichert.

Digitale Werbeträger sind derzeit noch nicht flächendeckend über ganz Deutschland verteilt, wie es bei klassischen Werbeträgern der Fall ist *(Fachverband Außenwerbung e.V., 2017*). Vielmehr konzentriert sich die überwiegende Anzahl von digitalen Werbeträgern räumlich auf die Großstädte von Deutschland. Daher werden auch nur die Textbeiträge abgefragt und gespeichert, welche in einer der 5 Städten mit der höchsten Abdeckung von digitalen Werbeträgern, gesendet wurden. Aus den, in *3.3 Stammdaten digitaler Werbeträger* beschrieben Daten, lässt sich ableiten, dass dies auf die folgenden Städte zutrifft: Berlin, Frankfurt am Main, Hamburg, Köln und München. Nach Sichtung der gesammelten Daten wird entschieden, welche dieser Städte in dieser Arbeit zur Analyse ausgewählt wird.

Die Textbeiträge wurden über einen Zeitraum von 9 Monaten gesammelt. In dieser Zeit konnten in der Summe 480.384 Textbeiträge gespeichert werden. Diese verteilen sich auf die 5 genannten Städte wie folgt:

- Berlin: 236.297
- Frankfurt am Main: 61.854
- Hamburg: 67.677
- Köln: 49.125
- München: 62.107

Die absolute Anzahl der für die Analyse verwertbaren Textbeiträge wird sich nach der Datenaufbereitung, wie in Kapitel 5 beschrieben, jedoch noch verringern.

3.2 Problematik der Datengewinnung

Wie durch eigene Analysen gezeigt hat, ist die Gewinnung der Daten mit ein paar Problemen bzw. Einschränkungen verbunden. Die absolute Anzahl von Textbeiträgen, die über die API abgefragt werden können, ist verhältnismäßig klein im Vergleich zu einem Land wie z.B. den

USA, in welchem in einer vergleichbar großen Stadt erheblich mehr Textbeiträge gesammelt werden können. Dies lässt sich dadurch begründen, dass in Deutschland das Datenschutzbewusstsein sehr hoch ist. Damit ein Textbeitrag über die API abgefragt werden kann ist es erforderlich, dass der Verfasser des Textbeitrages zuvor in seinem Profil eine entsprechende Einstellung vorgenommen hat, dass die gesendeten Textbeiträge für jeden öffentlich zugänglich sind. Die zweite Hürde besteht darin, dass ebenfalls eine Profileinstellung getroffen werden muss, die es gestattet zusätzlich die geographische Position des Benutzers zu speichern und publizieren. Vielen Benutzern in Deutschland ist Privatsphäre und Anonymität im Internet wichtig und sie deaktivieren daher die entsprechenden Einstellungen. Zu diesem Ergebnis kommt auch der EMC Datenschutzindex. Dies ist eine Studie, die 2014 mit 15.000 Verbrauchern in 15 Ländern durchgeführt wurde. Kernfrage der Studie war es herauszufinden, wie bereit Menschen sind auf Ihre Privatsphäre im Internet zu verzichten, Zugunsten von Bequemlichkeit und Vorteilen. Das Resultat dieser Studie ist ein Index, welcher die Einstellungen und Meinungen der befragten Verbraucher zum Thema Datenschutz und Privatsphäre im Internet wiederspiegelt. Deutschland belegte dabei den 15. Platz und ist somit, gemäß der Studie, das Land mit dem höchsten Datenschutzbewusstsein (*EMC Corporation, 2017*).

Ein weiteres Problem ist, dass sowohl die API von Twitter als auch die API von Flickr eine Limitierung vorsehen, wie viele Anfragen in einem definierten Zeitfenster an die API gesendet werden dürfen. Dies ist eine Maßnahme um den Datendurchsatz und die Serverlast zu begrenzen. Twitter erlaubt hier maximal 180 Anfragen in 15 Minuten (*API Rate Limits — Twitter Developers, 2017*). Im Gegensatz dazu erlaubt Flickr 3600 Anfragen in 60 Minuten. Diese Restriktion muss beim Sammeln der Daten beachtet werden. Bei Verstoß gegen diese Auflagen droht die Sperrung des Zugriffs auf die APIs (*Flickr: Der Flickr Entwickler-Leitfaden – API, 2017*).

Neben der Limitierung der Anzahl der Anfragen, muss bei der Twitter-API eine weitere Einschränkung beachtet werden. Die API liefert ausschließlich Daten, die maximal 7 Tage in der Vergangenheit liegen (*The Search API — Twitter Developers, 2017*). Um Twitter-Textbeiträge über einen längeren Zeitraum zu speichern ist es daher notwendig, die API

kontinuierlich in einem 7-Tage-Zyklus anzusteuern und die gesammelten Textbeiträge in einer Datenbank permanent zu speichern. Die Flickr-API hat in diesem Zusammenhang keine Restriktion und erlaubt Abfragen zu einem beliebigen Zeitraum in der Vergangenheit.

3.3 Stammdaten digitaler Werbeträger

Die digitale Media Institut (DMI) kooperiert mit Anbietern von digitalen Werbeträgern, die Werbeplatz auf ihren Screens vermieten. Das DMI bietet jedem Anbieter die technischen Möglichkeiten, ihre Stammdaten von digitalen Werbeträgern in Deutschland in eine zentrale Datenbank einzupflegen. Die Stammdaten umfassen neben den Standortinformationen u.a. auch Details über Buchungskonditionen, Kosten, verfügbare Zeitschienen usw. Eine genaue Beschreibung der Daten können über das DMI angefragt werden *(DMI Digital Media Institute UG, 2016*). Die Stammdaten werden für diese Arbeit von dem DMI zur Verfügung gestellt.

3.4 Data-Crawler

Wie in Kapitel *3.1 Eingrenzung der Zieldaten* und Kapitel *3.2 Problematik der Datengewinnung* erläutert, bedarf es einer Automatisierung um die Textbeiträge der Twitter-API und Flickr-API zu sammeln und zu speichern. Dies geschieht mit Hilfe eines in Python geschriebenen Skripts, welches im Folgenden als *Data-Crawler* bezeichnet wird. Eine Basis-Version des Data-Crawlers wurde von der Universität Salzburg für diese Arbeit zur Verfügung gestellt. Um den beschriebenen Anforderungen gerecht zu werden, ist der Data-Crawler leicht modifiziert worden. Die Arbeitsweise des *Data-Crawlers* ist in Abbildung 4 visualisiert. Der Data-Crawler läuft alle Punkte einer Bounding-Box ab und ermittelt zu der aktuellen geographischen Position die Textbeiträge der jeweiligen API. Die erhaltenen Textbeiträge werden in einer PostgreSQL-Datenbank gespeichert. Dieser Prozess wird solange wiederholt, bis alle Positionen der aktuellen Bounding-Box nach neuen Textbeiträgen überprüft worden sind. Anschließend wird der gleiche Prozess für die nächste Bounding-Box wiederholt. Das Skript stellt bei diesen Prozessen sicher, dass alle vorangehenden beschriebenen Restriktionen berücksichtigt werden. Darüber hinaus wird

sichergestellt, dass weder Dubletten, noch Textbeiträge ohne Koordinaten in der PostgreSQL-Datenbank gespeichert werden.

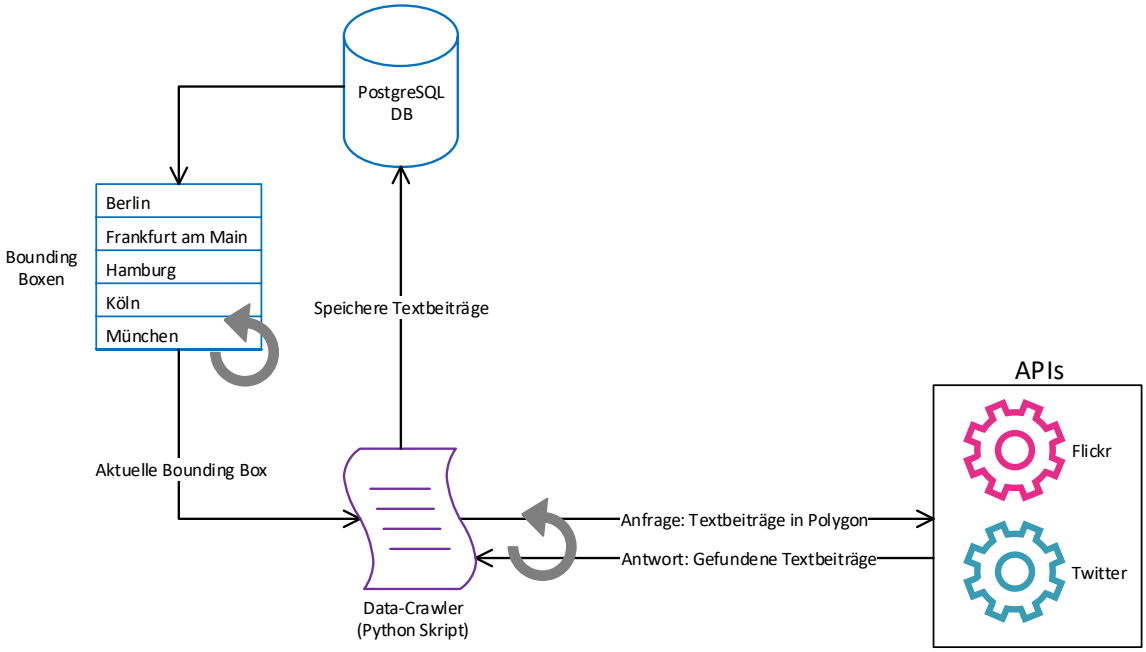


Abbildung 4: Prozesse - Data-Crawler

4. Systemarchitektur

4.1 Übersicht

Abbildung 5 zeigt einen Überblick aller Komponenten, die für die Bildung der thematischen Cluster in dieser Arbeit angewendet werden. Die hier dargestellten Komponenten werden in den folgenden Kapiteln *4.2 Datenbank* und *4.3 Programmiersprachen und Bibliotheken* beschrieben.

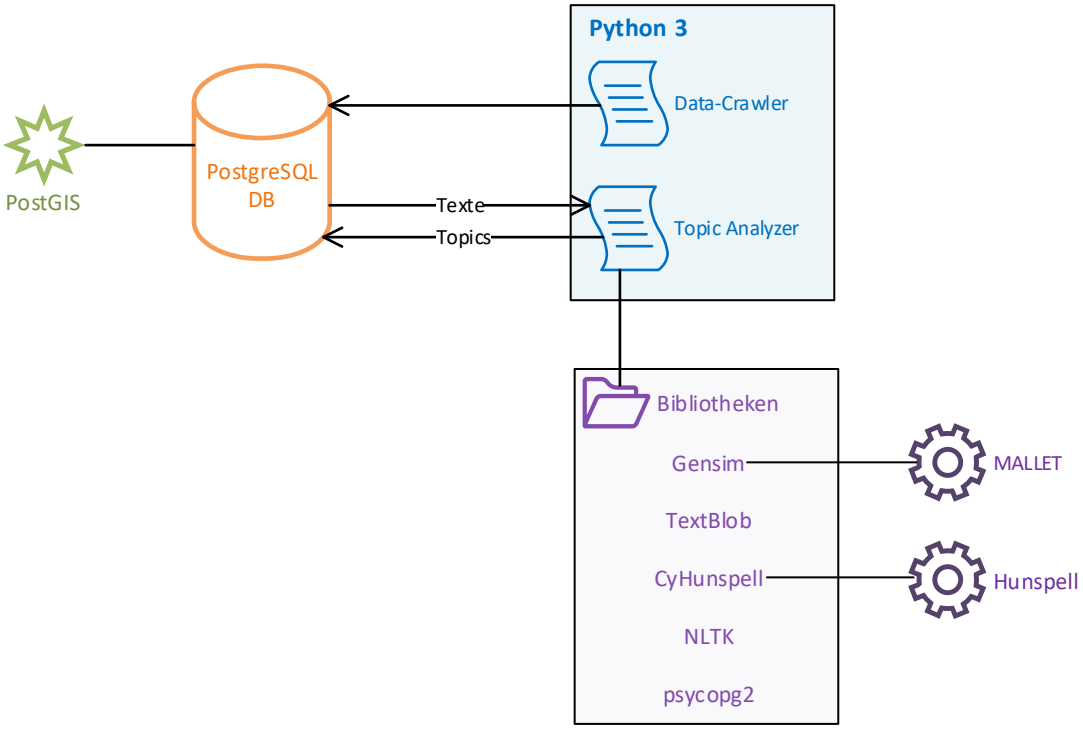


Abbildung 5: Übersicht Systemarchitektur

4.2 Datenbank

Die Speicherung aller datenbankseitigen Informationen erfolgt in einer PostgreSQL-Datenbank. PostgreSQL ist eine objekt-relationale Datenbank, welche von einer Open-Source-Community entwickelt wird (*PostgreSQL: About, 2017*). Aufgrund des freien Zugang, gibt es viele Drittanbieter die Datenbank-Komponenten oder -Erweiterungen entwickeln welche ebenfalls frei zugänglich sind. Eine dieser Datenbank-Erweiterungen ist PostGIS und

kommt in dieser Arbeit zum Einsatz. Die Erweiterung ermöglicht es Geometrische-Daten in einer PostgreSQL-Datenbank zu verwalten. Neben dem speziellen Datentyp „Geometry“ bietet PostGIS eine umfangreiche Funktions-Bibliothek, welche die Verarbeitung und Analyse von Geometrien ermöglichen (*PostGIS, 2017*).

4.3 Programmiersprachen und Bibliotheken

Das Script zur Extraktion der Themen, im Folgenden „Topic Analyzer“ genannt, ist in Python 3 entwickelt. Die Wahl fällt auf diese Programmiersprache, da Python eine sehr große Community hat, welche viele Bibliotheken veröffentlicht und zur freien Nutzung anbietet, welche in dieser Arbeit genutzt werden können. Darüber hinaus bestehen bereits erste Erfahrungen im Umgang und der Entwicklung mit Python.

Eine wichtige Entscheidung betrifft die Wahl der Bibliothek, mit welcher die eigentlichen LDA-Berechnungen durchgeführt werden. Hier gibt es mehrere Bibliotheken, die einen LDA-Algorithmus implementiert haben. Die Wahl fällt in dieser Arbeit auf Bibliothek ***gensim*** (*gensim: topic modelling for humans*, 2017) in Kombination mit dem Java-basierten Toolkit ***Mallet*** (*MALLET homepage*). Die Bibliothek gensim hat bereits einen eigenen Algorithmus für LDA-Topic-Models integriert, jedoch hat das Toolkit Mallet in selbst durchgeführten, vorangehenden Test die besseren Ergebnisse geliefert. Um Mallet in einer Python-Anwendung nutzen zu können, ist ein Wrapper notwendig, der die Anweisungen zwischen Java und Python übersetzt. Gensim hat solch einen Wrapper implementiert der es ermöglicht das Toolkit Mallet zu starten und die Ergebnisse auszuwerten. (*gensim: topic modelling for humans*, 2014). Mallet hat in dem LDA-Algorithmus bereits Gibbs-Sampling integriert, um die Themen-Wahrscheinlichkeiten anzunähern.

Damit das Script die Daten aus der PostgreSQL-Datenbank in den Arbeitsspeicher laden kann und ebenso die ermittelten Ergebnisse in die Datenbank zurück schreiben kann, wird die Bibliothek ***psycopg2*** (*PostgreSQL + Python | Psycopg*) eingesetzt. Mit psycopg2 kann sowohl die Verbindung zur Datenbank hergestellt werden, als auch selektierende bzw. manipulierende Abfragen durchgeführt werden.

Für die Datenaufbereitung kommen mehrere Bibliotheken zum Einsatz. Die genauen Prozesse werden in Kapitel *5. Datenaufbereitung* näher beschrieben. Die Datenaufbereitung beinhaltet eine Rechtschreibkorrektur. Für diese Aufgabe wird die Software ***Hunspell*** (*Hunspell: About, 2017*) verwendet, welche in C++ geschrieben ist und von vielen großen Firmen wie Google oder Mozilla eingesetzt wird. Damit auf Hunspell aus Python zugegriffen werden kann, ist hier ebenfalls ein Wrapper notwendig, welcher den Aufruf von Hunspell ermöglicht. Dies wird in dieser Arbeit mit der Python-Bibliothek ***CyHunspell*** (*CyHunspell 1.2.0 Python Package Index, 2017*) erreicht.

Eine weitere Python-Bibliothek, welche bei der Datenaufbereitung zum Einsatz kommt, ist ***NLTK*** (Natural Language Toolkit) (*Natural Language Toolkit — NLTK 3.2.4 documentation*, 2017). NLTK ist eine Python-Bibliothek, welche für Text- bzw. Sprachverarbeitung eingesetzt wird. Neben NLTK wird in dieser Arbeit auch ***TextBLOB*** (*TextBlob: Simplified Text Processing — TextBlob 0.13.0 documentation*, 2017) eingesetzt, welche ähnliche Funktionen anbietet. Es ist notwendig NLTK und TextBLOB zu kombinieren, da die unterschiedlichen Sprachen der gesammelten Textbeiträge unterschiedliche Probleme mit sich bringen.

5. Datenaufbereitung

Um die Ergebnisse der zu extrahierenden Themen zu optimieren, welche mit Hilfe von LDA ermittelt werden sollen, werden die Textbeiträge zunächst aufbereitet und für die weitere Verarbeitung optimiert. Dazu durchlaufen die Textbeiträge mehrere Prozesse, welche in diesem Kapitel aufgezeigt werden.

5.1 Grundlegende Aufbereitungen

Da LDA sprachunabhängig funktioniert ist es besonders wichtig zu regulieren, welche Texte als Eingabeparameter definiert werden. Zunächst ist es wichtig, dass Zeichenketten, welche sich häufig wiederholen aber keinen inhaltlichen Mehrwert liefern, zu entfernen. Die betrifft vor allem URLs, welche in den Textbeiträgen eingebettet sind. Um die URLs zu identifizieren, werden reguläre Ausdrücke verwendet.

Um ein homogenes Ergebnis zu erzielen, werden alle Zeichen eliminiert, welche weder im lateinischen Alphabet vorkommen, noch einen Umlaut oder ein „ß“ darstellen. Dies beinhaltet ebenfalls die Entfernung aller Zeilenumbrüche. Ebenfalls werden alle Worte entfernt deren Länge kleiner oder gleich 2 sind. Diese Wörter sind mit hoher Wahrscheinlichkeit nur Füllwörter und würden das Ergebnis verfälschen.

5.2 Blacklist

Erste Testdurchläufe haben gezeigt, dass nahezu alle Themen geprägt von Wörtern mit lokalen Bezug sind. So bestanden die ersten ermittelten Themen in Köln fast alle aus Wörtern wie „Köln“, „Nordrhein“, „Rhein“, „Westphalen“, „Deutschland“, „Germany“, „Cologne“, usw. Dies führt dazu, dass keine verwendbaren Zielgruppen-Themen ermittelt werden können. Zwar wurden diese Wörter tatsächlich gepostet, jedoch machen sie es unmöglich verwertbare Ergebnisse zu ermitteln. Aus diesem Grund wird manuell eine Blacklist erstellt. Alle Wörter die auf dieser Liste stehen werden als „unerwünscht“ definiert und aus allen Textbeiträgen entfernt. Die Entscheidung, welche Wörter der Blacklist

hinzugefügt werden, geschieht durch schrittweise Annäherung. So wurden in mehreren Durchläufen die Themen analysiert, unerwünschte Wörter in den resultierenden Themen identifiziert und anschließend der Blacklist hinzugefügt.

5.3 Bot-Beiträge

Ziel dieser Arbeit ist es, wie eingehend erläutert, Zielgruppen aus Textbeiträgen sozialer Medien ableiten zu können. Dabei ist der Kerngedanke, dass diese Textbeiträge von Menschen verfasst wurden, welche die Zielgruppen repräsentieren. Jedoch gibt es auch Textbeiträge, die nicht von Menschen, sondern von Programmen – sogenannten Bots – verfasst werden. Dies ist möglich, da die APIs der hier verwendeten Plattformen Twitter und Flickr nicht nur das „Lesen" ermöglichen, sondern auch das „Schreiben" von Textbeiträgen. Diese Bots verfassen in der Regel kontinuierlich Nachrichten mit ähnlichen Inhalten, die die hier ermittelten Ergebnisse verfälschen würden. So gibt es z.B. einen Bot im Raum Köln, welcher jede halbe Stunde das aktuelle Wetter in Köln und die Windgeschwindigkeit postet. Da diese Beiträge einen verhältnismäßig hohen Anteil aller Textbeiträge in Köln einnehmen, ist die Wahrscheinlichkeit hoch, das eines der resultierenden Themen, das Wetter sein wird. Dies bedeutet aber nicht, dass Kölner ein hohes Interesse an dem Wetter haben. Daher ist es wichtig, diese Bot-Beiträge zu identifizieren und aus dem Pool aller Textbeiträge zu löschen. Bei der Entwicklung eines Ansatzes zur Identifizierung dieser Bot-Beiträge, wurde zunächst versucht die Bots mit Hilfe der Korrelation zu finden. Dies geschah unter der Annahme, dass Textbeiträge von Bots immer ähnlich im Satzaufbau und in der Wortwahl sind, wie z.B.

#Köln 18.05 09:20 Temperatur 20C leichte Schauer Wind N 6 km/h

#Köln 18.05 05:20 Temperatur 19C trocken Wind O 2 km/h

#Köln 18.05 02:20 Temperatur 18C gering bewölkt / klar Wind SW 9 km/h

Dies hat sich jedoch als nicht praktikabel erwiesen, da viele Bot-Beiträge oft nur an einem einzigen Wort zu erkennen sind. So gibt es z.B. einen Bot, welcher Stellenausschreibungen postet und dies nur an dem Wort „Stellenausschreibung" zu erkennen ist. Daher wird hier

ein Ansatz gewählt, welcher die Worthäufigkeit aller Textbeiträge eines Benutzers ermittelt und diese in das Verhältnis aller geposteten Textbeiträge stellt. Wenn unverhältnismäßig viele Textbeiträge eines Benutzers immer wieder das gleiche Wort beinhalten, ist dieser Benutzer mit hoher Wahrscheinlichkeit ein Bot und seine Textbeiträge werden aus dem Pool aller Textbeiträge gelöscht. Da bei diesem Vorgehen auch Beiträge gelöscht werden würden, welche z.B. viele URLs beinhalten oder Füllwörter wie „und“ wird dies erst durchgeführt, nachdem die Texte wie in Kapitel *5.1 Grundlegende Aufbereitung* und Kapitel *5.2 Blacklist* beschrieben, bearbeitet worden sind.

Zusätzlich werden alle Textbeiträge entfernt, welche von einem Benutzer mindestens 3-mal in exakt demselben Wortlaut gepostet wurden. Besonders bei langen Textbeiträgen ist es unwahrscheinlich, dass ein Benutzer mehrmals das gleiche schreibt. Diese Nachrichten lassen eine automatisierte Entstehung vermuten und werden daher gelöscht.

5.4 Spracherkennung

Obwohl, oder gerade weil, die Textbeiträge ausschließlich in den Großstädten Deutschlands gesammelt werden, sind die gesammelten Textbeiträge nicht ausschließlich in deutscher Sprache verfasst. Die 5 ausgewählten Städte sind beliebte Reiseziele, was dazu führt, dass viele Touristen dort auch Textbeiträge in ihrer eigenen Sprache posten. Nach Sichtung der gesammelten Daten hat sich gezeigt, dass der überwiegende Anteil der Textbeiträge sowohl in Deutsch, als auch in Englisch verfasst ist. Um hier eine möglich hohe Anzahl an verwertbaren Textbeiträgen zu erhalten, werden daher alle Textbeiträge zur Analyse herangezogen, die in diesen beiden Sprachen verfasst worden sind. Dies ist möglich, da LDA sprachunabhängig funktioniert und somit Themen in Texten unterschiedlicher Sprachen bilden kann. Das Vorgehen hat jedoch zur Folge, dass die Sprache eines jeden Textbeitrages identifiziert werden muss um zu entscheiden, ob er gelöscht oder gehalten werden soll. Darüber hinaus ist es wichtig zu definieren, in welcher Sprache ein Textbeitrag geschrieben ist, da die weiteren Verarbeitungsschritte wie z.B. Rechtschreibkorrektur oder Lemmatisierung nur in Kombination mit dem „richtigen“ Wörterbüchern erfolgreich durchgeführt werden können. Um die Sprache eines Textbeitrages zu bestimmen, wird die

Python-Bibliothek *TextBLOB* verwendet. Diese hat eine Spracherkennungs-Funktion implementiert, mit welcher ein übergebener Text sprachlich klassifiziert wird. In Abhängigkeit der gefunden Sprache wird anschließend entschieden, wie mit dem Text weiter verfahren wird.

5.5 Rechtschreibkorrektur

Alle Wörter aller Textbeiträge durchlaufen vor der Themenextraktion mit LDA eine Rechtschreibkorrektur. Dies hat das Ziel, die Ergebnisse der resultierenden Themen zu verbessern. Wenn dasselbe Wort in unterschiedlichen Textbeiträgen einmal korrekt und einmal falsch geschrieben ist, werden sie dennoch wie zwei unterschiedliche Wörter behandelt. Dies ist bereits der Fall, wenn sich die beiden Wörter nur in einem einzigen Buchstaben unterscheiden. Für die Rechtschreibkorrektur wird *Hunspell* verwendet (siehe Kapitel *4.3 Programmiersprachen und Bibliotheken*). Um *Hunspell* in Python verwenden zu können wird mit der Python-Bibliothek *Cyhunspell* gearbeitet. Alternativ kann hier auch mit der Python-Bibliothek *enchant* gearbeitet werden. Diese unterstützte jedoch zum Zeitpunkt der Erstellung dieser Arbeit nicht Python 3.

Hunspell muss in Verbindung mit einem Wörterbuch der entsprechenden Sprache angewendet werden. Die Wörterbücher die hier genutzt werden sind OpenSource und werden unter anderem in *LibreOffice* und *Mozilla Firefox* verwendet (*LibreOffice/dictionaries*). Die Wörterbücher sind Textdateien, welche Wörter und deren unterschiedlichen Schreibregeln beinhalten. Es wird jeweils ein Wörterbuch für die englische Sprache und ein Wörterbuch für die deutsche Sprache eingebunden. In Abhängigkeit der identifizierten Sprachen der Textbeiträge, wird die Rechtschreibkorrektur mit dem jeweils entsprechenden Wörterbuch durchgeführt. Kann ein Text keiner Sprache zugeordnet werden, wird er unbehandelt an den nächsten Text-Aufbereitungsschritt übergeben.

5.6 Stoppworte

Stoppworte tragen keine wichtige, themenbildende Information und haben daher keine Bedeutung für den Inhalt einer Textnachricht. Stoppwörter sind z.B. Füllwörter oder Artikel wie „und“, „in“, „somit“, „der“, „die“, usw. Da diese Wörter jedoch häufig in einem Text vorkommen, werden sie die Ergebnisse für die resultierenden Themen verschlechtern und sollten daher gelöscht werden (*Rajaraman et al., 2014*). Das Identifizieren von Stoppwörtern wird in dieser Arbeit mit Hilfe der Python-Bibliothek *NLTK*, sowie einer Stoppwort-Liste der jeweiligen Sprache durchgeführt. Wird ein Wort als Stoppwort erkannt, wird es aus dem Textbeitrag entfernt. Da die Textnachrichten in unterschiedlichen Sprachen verfasst sind, werden die Stoppwörter der vier Sprachen entfernt, in denen die meisten Texte verfasst sind. Dies sind: Deutsch, Englisch, Französisch, Türkisch und Spanisch.

5.7 Lemmatisierung

Ein Problem bei der Extraktion der Themen mit Hilfe von LDA ist, dass ein Wort in vielen Wortformen vorkommen kann. Der LDA-Algorithmus, interpretiert diese unterschiedlichen Wortformen nicht als ein gemeinsames Wort, sondern als viele unterschiedliche Wörter. Die Wörter „laufen“, „laufe“ und „lauf“ werden nicht automatisch mit dem Wort „laufen“ in Kontext gebracht, sondern als drei unterschiedliche Wörter behandelt. Um hier ein besseres Ergebnis zu erzielen, werden alle Worte vor der Themenextraktion aufbereitet, so dass sie als das Wort ihrer Grundform behandelt werden. Um dies zu erreichen gibt es zwei gängige Verfahren: Stemming und Lemmatisierung. In einem ersten Testdurchlauf wurden die Worte mittels Stemming aufbereitet. Diese Methode verkürzt die Worte nach definierten Regeln. Dabei können die Themen zwar in einer guten Perfomance ermittelt werden, die resultierenden Ergebnisse sind jedoch nicht sehr gut. Viele Worte können durch die Wortverkürzung nur schwer auf dessen ursprüngliche Worte zurückgeführt werden und es ist kein deutlicher Zusammenhang zwischen den geclusterten Worten erkennbar. Besonders in der deutschen Sprache ist dieses Verfahren nicht immer von Erfolg gekrönt. Die Worte „gehst“, „ging“ und „gegangen“ können mit Stemming nur schwer auf das gemeinsame Wort „gehen“ zurückgeführt werden. Auch gibt es im Deutschen viele Worte, die sich aus

mehreren Worten zusammensetzten. Worte wie „Hausflur“ oder „Tischfußball“ bereiten Stemming-Algorithmen in der Regel Probleme. Dennoch sind die Ergebnisse in diesem Testdurchlauf besser, als eine Themenextraktion ohne Stemming *(Lewandowski, 2005)*.

Alternativ zu der Aufbereitung mit Stemming wurde in einem zweiten Test eine Aufbereitung mittels Lemmatisierung durchgeführt. Dieses Verfahren ist deutlich zeitaufwendiger, da hier mit Hilfe von Lexika und morphologischen Regeln die Worte auf ihre Grundform zurückgeführt werden (*Lewandowski, 2005*) . Die Ergebnisse, die hier mit Hilfe von Lemmatisierung ermittelt wurden, sind deutlich besser. Daher werden in dieser Arbeit die Textbeiträge mittels Lemmatisierung aufbereitet.

Ähnlich wie bei der Durchführung der Rechtschreibkorrektur, muss bei jedem Textbeitrag entschieden werden, ob die Lemmatisierung mit Regeln der deutschen Sprache oder der englischen Sprache durchgeführt wird. Wird der Textbeitrag als englischer Text erkannt, wird die Lemmatisierung mit Hilfe der Python-Bibliothek *TextBLOB* durchgeführt. *TextBLOB* bietet jedoch nicht direkt eine Methode zur Lemmatisierung von deutschen Texten. Stattdessen wird die Python-Bibliothek *TextBLOB-DE* verwendet, welche auf *TextBLOB* basiert und Python 3 unterstützt (*textblob-de 0.4.2 Python Package Index, 2017*). Kann ein Text keiner Sprache zugeordnet werden, wird er unbehandelt an den nächsten Text-Aufbereitungsschritt übergeben.

5.8 Tokenization und Bag-of-Word

Wie in Kapitel *5.1 Grundlegende Aufbereitungen* beschrieben, werden alle Textbeiträge dahingehendend aufbereitet, dass nur noch Zahlen, Buchstaben und Leerzeichen in den Textbeiträgen enthalten sind. Bevor die Textbeiträge als Eingabeparameter dem LDA-Algorithmus übergeben werden, werden alle Textbeiträge und deren Worte in eine zweidimensionale Liste überführt. Dabei repräsentiert jede Zeile der Liste einen Textbeitrag. Jeder Textbeitrag wird nach Worten zerteilt und in einer untergeordneten Liste gespeichert. Als Trennzeichen wird jedes Leerzeichen innerhalb eines Textbeitrages definiert.

Diese zweidimensionale Liste wird mit Hilfe der Python-Bibliothek *gensim* in eine Bag-of-Word-Matrix umgewandelt. Diese Bag-of-Word-Matrix repräsentiert den letzten Aufbereitungsschritt der Textbeiträge, bevor die eigentliche LDA-Analyse mit *Mallet* durchgeführt wird.

5.9 Unicode-Emoticons

Nach Sichtung der gesammelten Textbeiträge fiel auf, dass viele Benutzer Emoticons verwenden, welche insbesondere von der Twitter-API in Unicode umgewandelt werden.

Emoticon		Unicode
😃	⇨	\ud83d\ude03
😇	⇨	\ud83d\ude07
😍	⇨	\ud83d\ude0d

Dies hat zur Folge, dass die Texte viele Wörter beinhalten, welche auf den ersten Blick sinnlos erscheinen und die Themengewinnung nur unnötig erschweren. Daher werden die Unicode-Zeichenketten durch Namen oder Worte ersetzt, welche das ursprüngliche Emoticon beschreiben. Die Beschreibungen werden einer frei zugänglichen Unicode-Liste entnommen (*wooorm/gemoji, 2017*).

Unicode		Beschreibung
\ud83d\ude03	⇨	happy; joy; laugh; pleased; smile
ud83d\ude07	⇨	innocent; angel
\ud83d\ude0d	⇨	love; crush; heart_eyes

5.10 Implementierung

Abbildung 6 zeigt die Reihenfolge der einzelnen Bearbeitungsschritte, in welcher die Daten für die weitere Analyse mit LDA aufbereitet werden. Die Implementierung sieht im ersten Schritt vor, dass die Daten zur Aufbereitung aus der PostgreSQL-Datenbank selektiert werden. Dazu wird im Vorfeld definiert, welche Daten analysiert werden sollen. Mittels der fest übergebenen Bounding-Box-ID und dem gewünschten Zeitintervall wird automatisiert eine Tabelle aus dem gesamten Datenbestand aller gesammelten Textbeiträge generiert. Diese Tabelle dient als Quelltabelle für alle weiteren Bearbeitungsschritte.

Zunächst werden mit Hilfe von mehreren SQL-Update-Befehlen und Regular Expressions alle Zeichen und Zeichenketten eliminiert, welche in Kapitel *5.1 Grundlegende Aufbereitungen* beschrieben sind. Ebenfalls werden auf dem gleichen Weg alle Wörter in den Textbeiträgen gelöscht, welche zuvor auf der, in Kapitel *5.2 Blacklist* beschriebenen, Blacklist definiert worden sind. Dabei können Zeilen mit leeren Textbeiträgen entstehen, welche danach ebenfalls gelöscht werden.

Anschließend werden, wie in Kapitel *5.9 Unicode-Emoticons* beschrieben, alle Unicode-Zeichenketten durch beschreibende Wörter ersetzt die das entsprechende Emoticon repräsentieren.

Nachdem alle unerwünschten Zeichenketten entfernt worden sind, kann damit begonnen werden potentielle Bot-Beiträge zu identifizieren. Dazu werden, wie in Kapitel *5.3 Bot-Beiträge* beschrieben, alle Benutzer, deren Textbeiträge in Folge dieses Prozesses markiert werden, in einer temporären Arbeitstabelle gesichert. Anschließend werden alle Textbeiträge dieser Benutzer aus der zu analysierenden Tabelle gelöscht.

Sobald alle Bearbeitungsschritte, die mit Hilfe von SQL durchgeführt werden können, abgeschlossen sind, wird dazu übergegangen die Daten aus der Tabelle der PostgreSQL-Datenbank zu selektieren und in den Arbeitsspeicher zu laden. Dazu werden alle

Textbeiträge wie in Kapitel *5.8 Tokenization und Bag-of-Word* beschrieben tokenisiert und in eine zweidimensionale Liste überführt.

Für jeden Textbeitrag wird eine Sprachidentifizierung durchgeführt (nachzulesen in Kapitel *5.4 Spracherkennung*). Weiterverarbeitet werden nur die Textbeiträge, welche als in Deutsch oder Englisch verfasst erkannt worden sind. Alle anderen Textbeiträge werden zwar nicht weiter aufbereitet, bleiben aber im Pool der zu analysierenden Textbeiträge. Alle Textbeiträge die einer der beiden Sprachen zugeordnet werden können, durchlaufen zunächst eine Rechtschreibkorrektur, wie sie in Kapitel *5.5 Rechtschreibkorrektur* beschrieben ist.

Nachdem alle möglichen Rechtschreibfehler so gut wie möglich korrigiert worden sind, wird im Anschluss je Wort eine Lemmatisierung durchgeführt und alle Stoppworte entfernt (siehe Kapitel *5.6 Stoppworte* und *5.7 Lemmatisierung*). Bevor die Textbeiträge der LDA-Analyse übergeben werden, wird die aufbereitete zweidimensionale Liste in eine Bag-of-Wort-Matrix überführt.

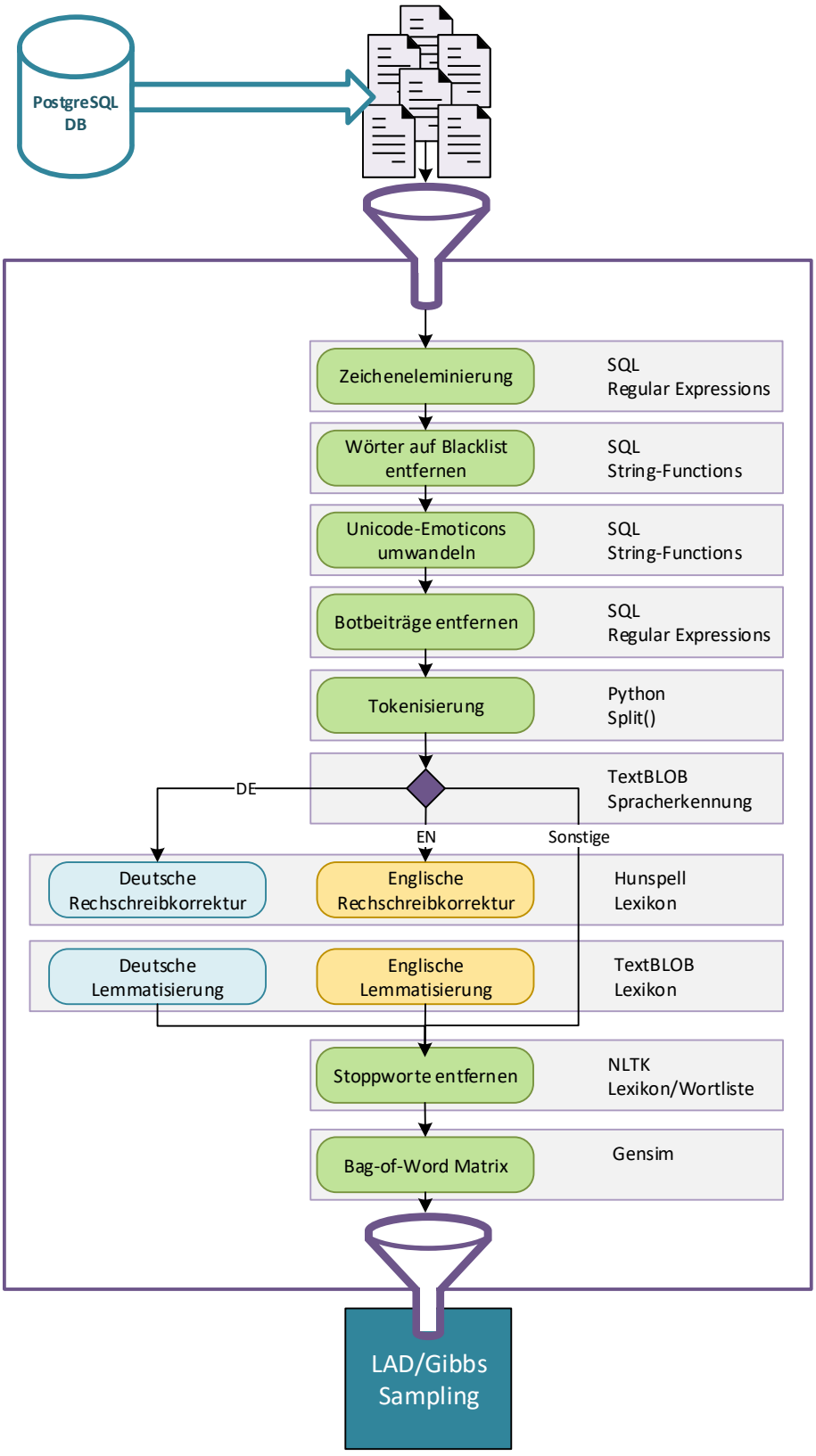


Abbildung 6: Prozesskette - Datenaufbereitung

6. Themenextraktion

6.1 LDA-Model

Das Resultat einer jeden LDA-Analyse mit dem Toolkit *Mallet* (siehe Kapitel *4.3 Programmiersprachen und Bibliotheken*) ist ein LDA-Model. Jedes ermittelte LDA-Model stellt ein komplexes Objekt dar, welches im Zuge der LDA-Analyse immer wieder aktualisiert wird und am Ende alle Wahrscheinlichkeiten der Dokument-Themenverteilung sowie die Wort-Themenzusammensetzungen hält. Damit das LDA-Model mit Hilfe von *Mallet* errechnet werden kann, werden der aufzurufenden Methode drei wichtige Übergabeparameter mitgegeben: Der zu analysierende Textkorpus in Form einer Bag-of-Word-Matrix, dem dazugehörigen Dictonary sowie die Anzahl der Themen, die ermittelt werden sollen.

Wie in Abbildung 7 zu sehen ist, kann die Bag-of-Word-Matrix nur erzeugt werden, wenn im Vorfeld das passende Dictonary ermittelt worden ist. Dies erklärt sich dadurch, dass die Bag-of-Word-Matrix hier die Information der Worthäufigkeiten in Form von ID-Verweisen hält. Welche ID welchem Wort zugewiesen ist, kann über das Dictonary zurückverfolgt werden. Um das Dictonary zu erzeugen, wird jedem Wort des gesamten Textcorpus eine ID zugewiesen. Dabei bekommt jedes Wort dokumentenübergreifend eine eindeutige ID. Sobald das Dictonary vorhanden ist, kann die Bag-of-Word-Matrix erstellt werden. Diese speichert für jedes Dokument die IDs der vorkommenden Wörter, sowie die Häufigkeit deren Auftretens innerhalb des Dokumentes (*gensim: topic modelling for humans*, 2017).

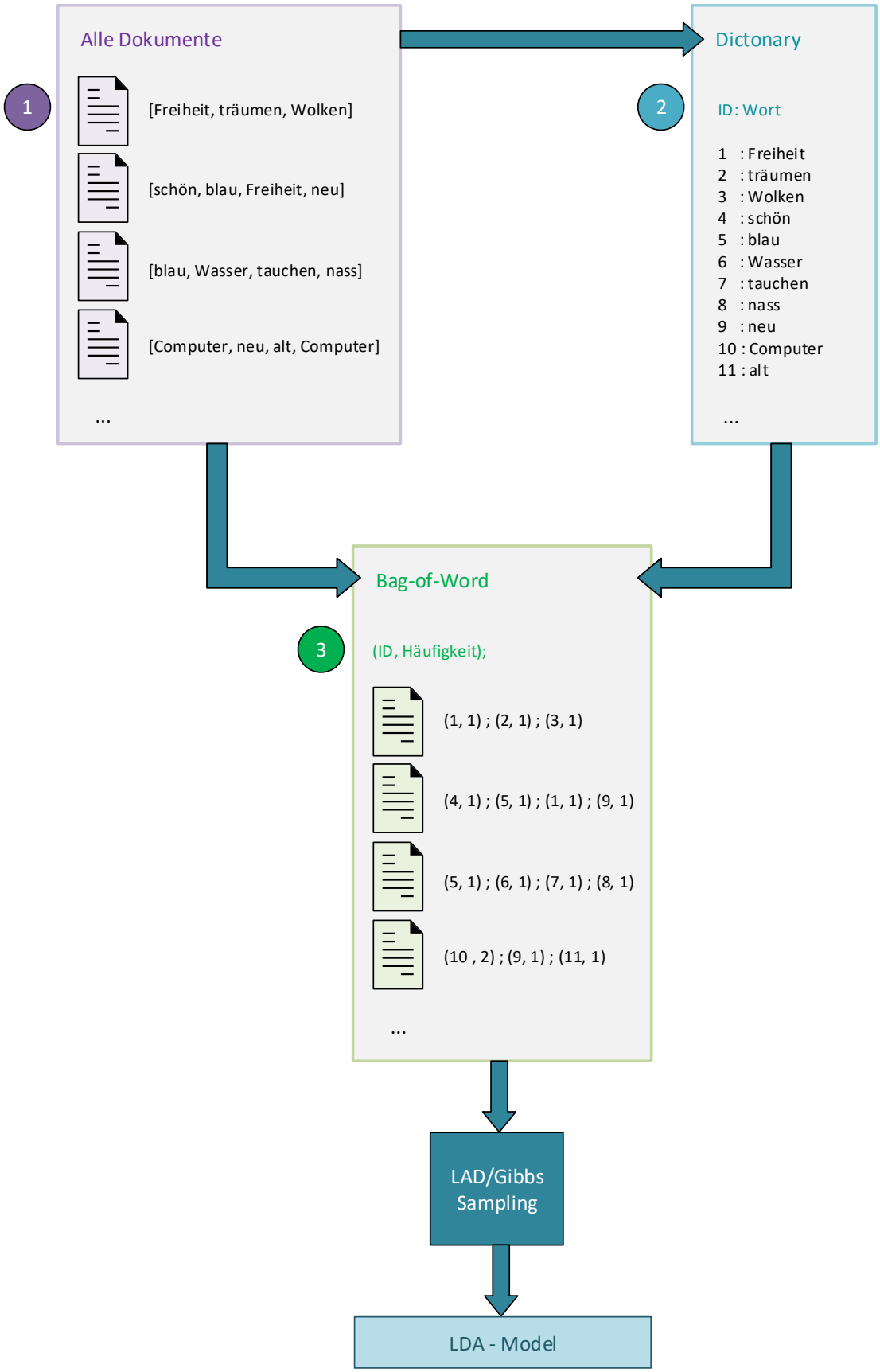


Abbildung 7: Generierung Dictonary/Bag-of-Word

6.2 Themenspeicherung

Ein zentraler Kernaspekt dieser Arbeit ist es, die Ergebnisse der LDA-Analyse kartografisch darzustellen und mit weiteren räumlichen Daten, wie den Standorten digitaler Werbeträger, in Kontext zu bringen. Neben der räumlichen Weiterverarbeitung, muss auch ein Rückschluss auf den Erstellungszeitpunkt möglich sein, um die Themen auch in Zeitschienen einordnen zu können. Die Liste, welche der LDA-Analyse übergeben wird, beinhaltet jedoch ausschließlich die weiterverarbeiteten Textbeiträge, welche schlussendlich als Bag-of-Word-Matrix übergeben werden. Um die hier gewonnen Ergebnisse der Themenextraktion wieder in Zusammenhang mit den ursprünglichen Daten bringen zu können, wird mit dem Index der Pyhton-Liste gearbeitet. Abbildung 8 zeigt die hier beschriebene Vorgehensweise: Wie in Kapitel

5.10 Implementierung erwähnt, wird für jede LDA-Analyse eine neue Tabelle erstellt. Diese hält neben allen notwendigen Informationen wie Text, Geometrie und Zeitstempel auch die ursprüngliche Message-ID, eine neue ID (Analysation-ID), welche alle zu analysierenden Textbeiträge aufsteigend mit einem Integer-Wert versieht. Die Analysation-ID ist lückenlos fortlaufend und wird erst vergeben, sobald alle datenbankseiteigen Textauf-bereitungsprozesse abgeschlossen sind. Die Selektion der Textbeiträge erfolgt nach der Sortierung der Analysation-ID. Die Reihenfolge der Textbeiträge der Python-Liste entspricht dieser Sortierung. Das LDA-Model, welches nach der LDA-Analyse zur Verfügung steht, hält zwei wichtige Ergebnis-Listen. Die erste Liste gibt an, mit welcher Wahrscheinlichkeit ein Wort zu einem Thema gehört. Die zweite Liste sagt aus, mit welcher Wahrscheinlichkeit ein Dokument einem Thema angehört. Die Ergebnisse dieser Liste sind in genau derselben Reihenfolge hinterlegt, wie die eingehend übergebene Python-Liste. Dies bedeutet, dass die Ergebnisse eines jeden Dokumentes, mit dem Zeilenindex der übergebenen Python-Liste übereinstimmen und somit auch der Analysation-ID. Jede der beiden Listen wird in einer CSV-Datei aufbereitet und jeweils in eine strukturgleiche Tabelle der Datenbank importiert. Die Reihenfolge der Dokumente wird wiederum als Analysation-ID abgespeichert. Dies stellt sicher, dass alle notwendigen Informationen für die weitere Auswertung mittels Tabellenverknüpfung herangezogen werden können.

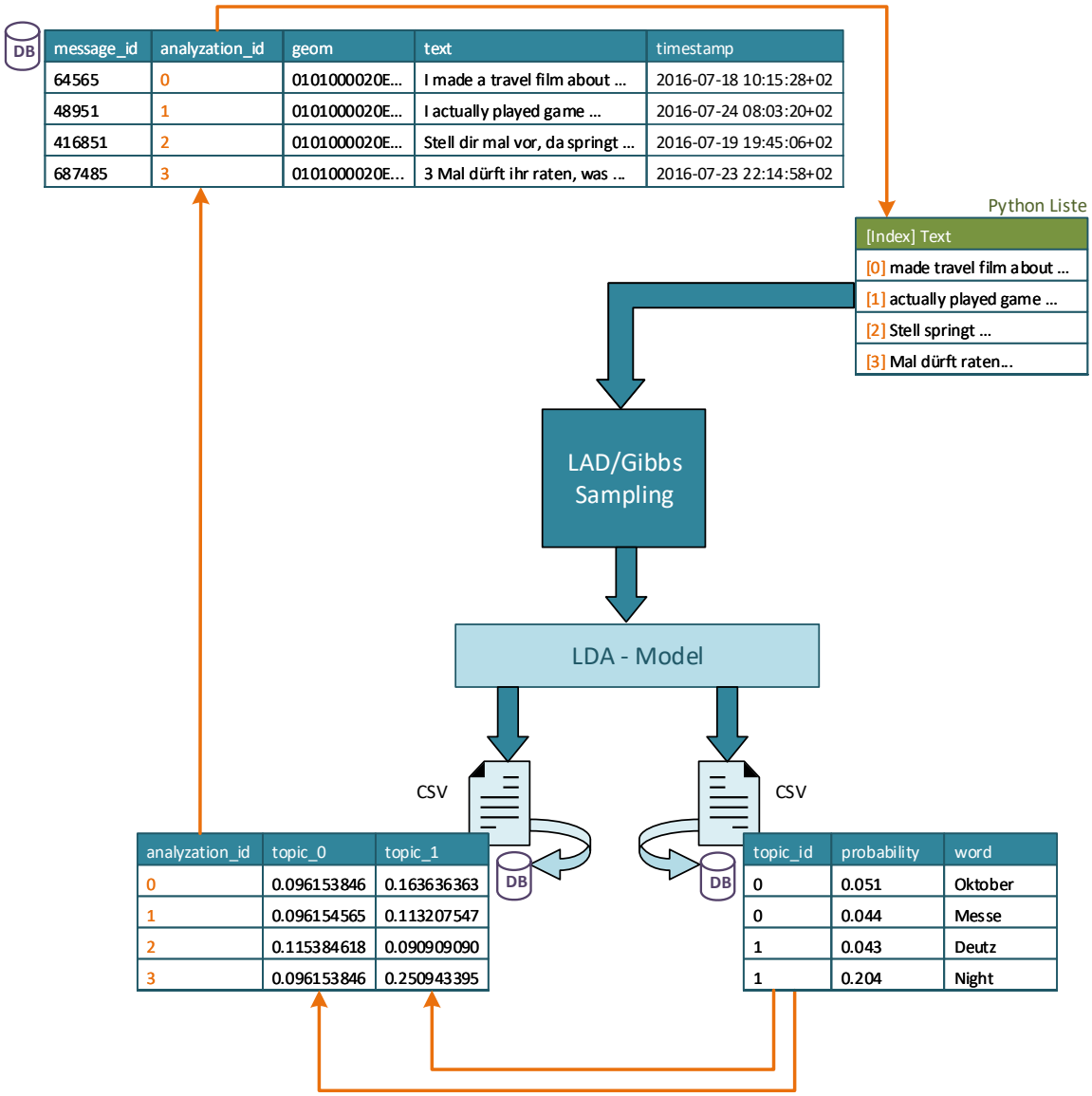


Abbildung 8: Speicherung Ergebnisse - LDA-Model

7. Evaluierung

7.1 Aufbau der Testdaten

Kernfrage dieser Arbeit ist es, wie eingehend erläutert, zu überprüfen, ob mit Hilfe von Daten aus sozialen Netzwerken und maschineller Themenextraktion Zielgruppen-informationen gewonnen werden können, die dabei helfen digitale Außenwerbung zu planen. Dazu werden mit Hilfe von LDA Themen in einer der 5 Städte Deutschlands extrahiert, in welchen die Textbeiträge gesammelt wurden. Nach Sichtung der Daten wird entschieden, dass für die Evaluierung Berlin am besten geeignet ist. In Berlin wurden mit Abstand die meisten Textbeiträge gesammelt. Ebenso findet man dort, im Vergleich zu den anderen vier Städten Frankfurt am Main, Hamburg, Köln und München, die größte Anzahl an digitalen Werbeträgern. Nachdem die Themen extrahiert worden sind, werden diese sowohl inhaltlich als auch räumlich ausgewertet und mit den Standorten der digitalen Werbeträger in Kontext gebracht.

7.1.1 Eingrenzung Flickr/Twitter

Wie in Kapitel *3. Datengrundlage* erläutert, wurden über einen Zeitraum von 9 Monaten die Textbeiträge der sozialen Netzwerke Twitter und Flickr gesammelt. Nach Sichtung der Daten hat sich jedoch gezeigt, dass die Daten der Flickr-API für diese Arbeit nur wenig bis gar nicht geeignet sind. Dies hat mehrere Gründe. Flickr ist im deutschsprachigen Raum nicht sehr populär. Demnach ist die Anzahl der auswertbaren Beiträge verhältnismäßig gering. Hinzu kommt, dass die wenigen gesammelten Beiträge nur schwer für diese Arbeit auswertbar sind. Flickr ist eine Plattform, deren primärerer Versendungszweck es ist, Fotos zu posten um sie mit der Community zu teilen (*Über Flickr, 2017*). Auswertbare schriftliche Inhalte sind nur vorhanden, wenn der Benutzer dem Foto einen erläuternden Titel hinzugefügt hat. Dies ist jedoch nur in den wenigsten Fällen gegeben. Viele Benutzer vergeben entweder gar keinen Titel oder belassen ihn bei dem Fotonamen. Dieser ist jedoch sehr oft ein automatisch generierter Name, welcher keinen nennenswerten Informationsgehalt beiträgt wie z.B. „IMG_1843.jpg“. Die vergebenen Tags sind häufig nur Ortsbeschreibend wie z.B. „Berlin“ oder „Nordrhein Westfalen“. Der Zeitaufwand um die gesammelten Flickr-

Textbeiträge aufzubereiten ist sehr hoch im Vergleich zu der zu erwartenden Ergebnismenge an auswertbaren Textbeiträgen. Daher werden hier nur die erhobenen Daten der Twitter-API herangezogen. Alle weiteren Ausführungen werden sich nur auf die Twitter-Textbeiträge beziehen.

7.1.2 Übersicht Testdaten

Die Aufbereitung der Textbeiträge sowie die Themenextraktion und deren Speicherung werden, wie in den Kapiteln *5. Datenaufbereitung* und *6. Themenextraktion* beschrieben, durchgeführt. Der Themengewinnung liegen folgende Daten zu Grunde:

Tabelle 2: Kenndaten gesammelter Tweets

	Berlin
Bounding Box	(13.05355, 52.330269) (13.72616, 52.667511)
Zeitspanne	01.08.2016 - 31.04.2017
Anzahl Tweets insgesamt	236.297
Anzahl Tweets aufbereitet	178.809
Themenanzahl	50

Im Anschluss an die Themenextraktion sollen die Textbeiträge mit den digitalen Werbeträgern räumlich verortet werden um eine mögliche Media-Empfehlung aussprechen zu können. Digitale Außenwerbeträger haben die Besonderheit, dass sie pro Zeitschiene gebucht werden können. Eine Zeitschiene ist ein definiertes Zeitfenster von 3 Stunden beginnend bei 00:00 Uhr. Um eine Aussage treffen zu können, welcher digitale Außenwerbeträger zu welcher Uhrzeit und passend zu einem Thema gebucht werden sollte, werden die Themen je Zeitschiene extrahiert. Mittels des Timestamps der aufbereiteten Textbeiträge werden die Daten in 8 Tabellen einsortiert. Die 50 Themen jeder Zeitschiene werden separat ermittelt.

Tabelle 3: Gesammelte Tweets je Zeitschiene

	Anzahl Tweets
00:00 – 03:00 Uhr	23.803
03:00 – 06:00 Uhr	10.903
06:00 – 09:00 Uhr	10.432
09:00 – 12:00 Uhr	30.599
12:00 – 15:00 Uhr	37.401
15:00 – 18:00 Uhr	27.984
18:00 – 21:00 Uhr	3.591
21:00 – 22:00 Uhr	34.096

7.1.3 Blacklist-Parameter

Nach den ersten Testdurchläufen der LDA-Analyse hat sich gezeigt, dass viele der gesendeten Textbeiträge einen Hashtag bzw. eine namentliche Nennung der Stadt beinhalten, in welcher sich die betreffende Person aufhält. Dies bedeutet, dass in Berlin sehr viele Textbeiträge die Worte „Berlin“ oder „Deutschland“ beinhalten. Dies hat zur Folge, dass diese Worte in vielen Themen als sehr themenbildend geclustert werden. In Hinblick auf die Fragestellung dieser Arbeit, ist dies jedoch nicht förderlich. Eine Werbekampagne deren Zielgruppendefinition auf Menschen ausgelegt ist, die ein erhöhtes Interesse an der Stadt Berlin haben, wird es mit hoher Wahrscheinlichkeit nicht geben. Daher werden die folgenden Begriffe auf die Blacklist gesetzt und somit vor der eigentlichen LDA-Analyse eliminiert (siehe Kapitel *5.2 Blacklist*)

["berlin", "germany", "deutschland", "potsdam", "brandenburg"]

7.2 Zielgruppen-Themen

Die Textbeiträge der 8 Zeitschienen aus Tabelle 3 werden wie beschrieben nacheinander aufbereitet und mit Hilfe von LDA analysiert. Die Ergebnisse sind jeweils 50 geclusterte Themen pro Zeitschiene. Für die Auswertung der Themen, werden jeweils die 5 Worte mit der höchsten Themen-Wahrscheinlichkeit betrachtet. Dabei fällt auf, dass nicht alle Themen bzw. Worte eines Themas dazu geeignet sind, zur Zielgruppendefinition herangezogen

werden zu können. In der Zeitschiene 18:00 – 21:00 Uhr sind z.B. folgende Worte zu einem Thema zusammengefasst worden:

0.136	Wall
0.094	East
0.085	Side
0.081	gallery
0.019	west

Diese lassen zwar eindeutig einen Zusammenhang erkennen der mit dem Oberbegriff „Berliner Mauer“ beschrieben werden könnte, jedoch stellt dieses Thema kein großes Potential dar, als Zielgruppendefinition einer Werbekampagne herangezogen werden zu können.

Ebenfalls ist auffällig, dass keine Themenbildung erkennbar ist, die eine Abhängigkeit der Tageszeit vermuten lassen könnte. Vielmehr finden sich die meisten Themen in unterschiedlichen Wortzusammensetzungen in nahezu allen Zeitschienen wieder. So finden sich Worte eines Themas die mit „Party“ in Verbindung gebracht werden können nicht, wie vielleicht zu erwartend, ausschließlich in den Abend- und Nachstunden wieder, sondern sind über alle Zeitschienen hinweg zu finden.

Aus diesem Grund werden die Themen aller Zeitschienen identifiziert, die sowohl ein hohes Potential bieten, Basis einer Zielgruppendefinition sein zu können, als auch über alle oder mehrere Zeitschienen hinweg unter einem Oberbegriff zusammengefasst werden zu können. Es sind 8 Oberbegriffe erkennbar, die diese Voraussetzungen erfüllen: *Essen, Mode/Fashion, Party, Musik/Konzert/Event, Fotographie/Art/Urban, Social Media, Sport treiben* und *Gute Laune/Fröhlich*. Die zusammengefassten Themen und ihre Worte mit den 5 höchsten Wahrscheinlichkeiten, sind in Abbildung 9 zu sehen. Die Abbildung zeigt jeden Oberbegriff mit seinen identifizierten Themen der jeweiligen Zeitschienen in einer eigenen Farbe.

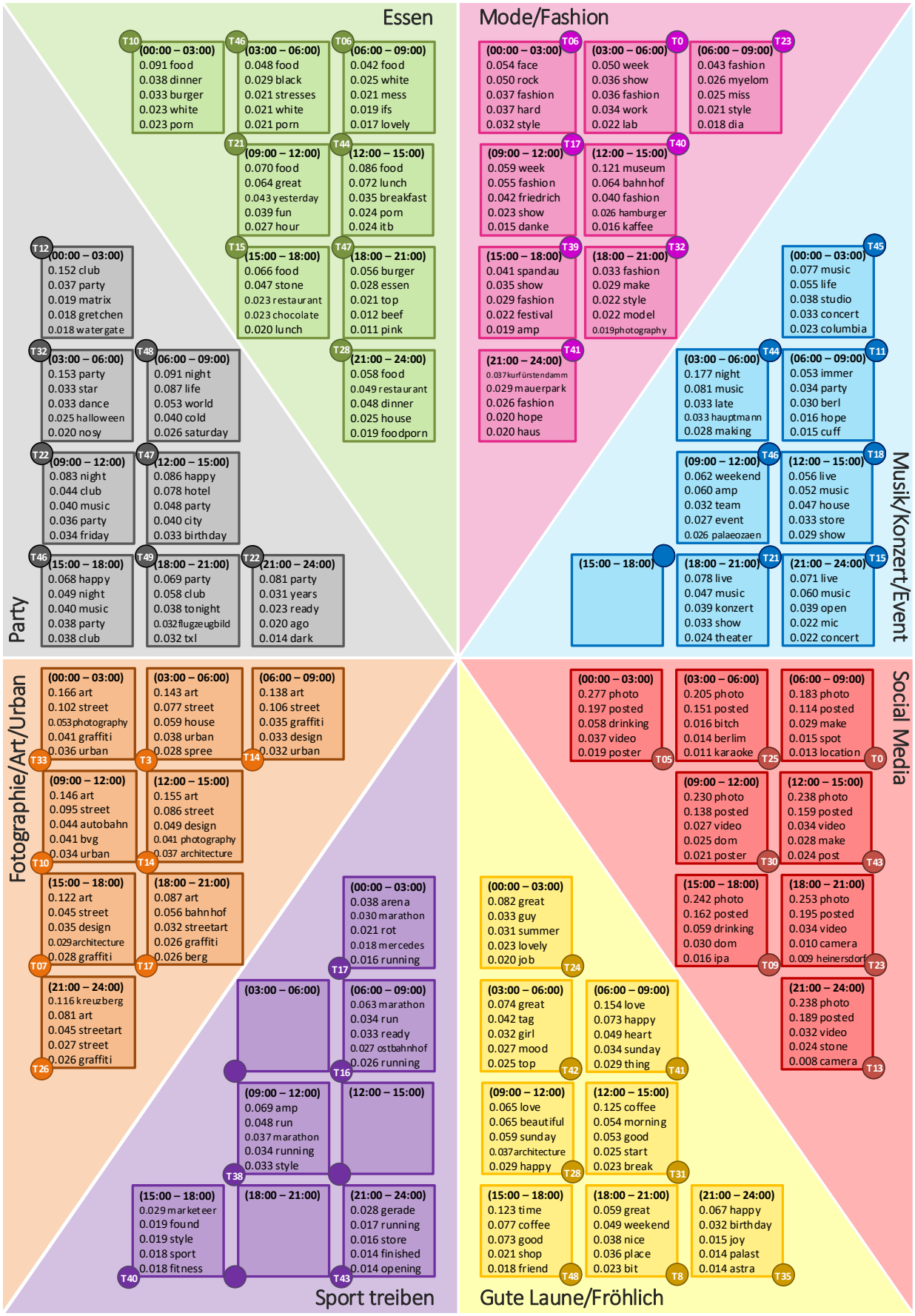


Abbildung 9: Themen sortiert nach Zeitschienen und zusammengefasst nach 8 Ober-Themen

7.3 Räumliche Auswertung Themen – Textbeiträge am Beispiel „Fashion/Mode“

Neben der inhaltlichen Auswertung der geclusterten Themen ist es Ziel dieser Arbeit zu überprüfen, ob die ermittelten Zielgruppen-Themen eine räumliche Auswertung erlauben, die zu einer möglichen Werbekampagne führen könnte. Die folgenden räumlichen Auswertungen werden mit der Software ArcMap der ArcGIS-Suite der Firma Esri durchgeführt. Dabei werden zwei Fragestellungen überprüft: Wie ist die räumliche Verteilung der Zielgruppen-Themen im geographischen Raum und welche Werbestandorte von digitalen Medien lassen sich daraus ableiten? Diese Fragestellungen werden exemplarisch an einem der 8 Oberbegriffe aus Abbildung 9, über alle Zeitschienen hinweg, überprüft. Die Wahl fällt hier auf die Textbeiträge aus Berlin, welche mit einer hohen Wahrscheinlichkeit dem Oberbegriff Mode bzw. Fashion zugeordnet werden können.

7.3.1 Interpolation

Um ein Bild davon zu bekommen, wie die räumliche Verteilung der Themen-Interesse Mode/Fashion in Berlin ist, werden die Punkt-Standorte der identifizierten Themen der jeweiligen Zeitschienen als Fläche interpoliert. Als Höhen-Wert der Interpolation wird die Wahrscheinlichkeit herangesogen, mit der eine Textnachricht dem betrachteten Thema angehört. Die Interpolationstechnik die hier angewendet wird ist die Inverse Distanzgewichtung (IDW). Abbildung 10 zeigt die Verarbeitungsabläufe die jeweils für die Textbeiträge/Tweets einer jeden Zeitschiene hierzu durchgeführt werden.

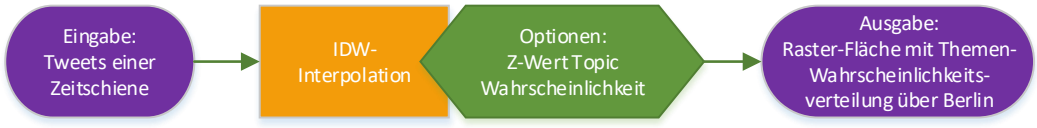


Abbildung 10: Abläufe in ArcMap zur Flächeninterpolation der Themen-Wahrscheinlichkeiten eines Themas und einer Zeitschiene

Die Interpolation wird für die folgenden Themen und Zeitschienen durchgeführt:

Tabelle 4: Mode/Fashion Themen der jeweilgen Zeitschienen

	Mode/Fashion: Thema
00:00 – 03:00 Uhr	T06
03:00 – 06:00 Uhr	T0
06:00 – 09:00 Uhr	T23
09:00 – 12:00 Uhr	T17
12:00 – 15:00 Uhr	T40
15:00 – 18:00 Uhr	T39
18:00 – 21:00 Uhr	T32
21:00 – 22:00 Uhr	T41

7.3.1.1 Inverse Distanzgewichtung (IDW)

Die Methode IDW ist eine lokale Interpolationstechnik. Sie basiert auf der Annahme, dass Punkte, die näher beieinander liegen sich ähnlicher sind als Punkte die weiter voneinander entfernt sind (Toblers 1. geographische Gesetz *(Tobler, 1970)*). Bei dieser Methode werden die Distanzen von den Messpunkten zu den Stützpunkten ermittelt. Der interpolierte Messwert ergibt sich aus dem gewichteten Mittelwert aller umliegenden Punkte. Je mehr Stützpunkte bei der Interpolierung angegeben werden, umso besser wird das zu erwartende Ergebnis (*Watson and Philip, 1985*).

7.3.1.2 Ergebnisse

Die Abbildung 11 - Abbildung 26 zeigen die Standorte aller verfassten Textnachrichten einer Zeitschiene und die jeweils dazu ermittelte Interpolationsfläche der Themen aus Tabelle 4. An den interpolierten Flächen ist zu erkennen, dass das Interesse an Mode/Fashion über die unterschiedlichen Zeitschienen hinweg räumlich in Bewegung ist. Es lassen sich jedoch nur schwer zentrale Ballungsgebiete erkennen. Die Flächen mit einem hohen Wahrscheinlichkeitswert sind oft sehr klein und in vielen Fällen nahezu homogen über ganz Berlin bzw. die Berliner Innenstadt verteilt. Um eine Aussage treffen zu können, welcher digitale Werbeträge idealerweise belegt werden sollte um mit einer hohen Wahrscheinlichkeit Personen zu erreichen, welche ein erhöhtes Interesse an Mode haben,

ist die Interpolationsfläche nicht gut geeignet. Jedoch lassen sich hiermit erste Tendenzen ermitteln und die Flächen zeigen deutlich, dass je Zeitschiene eine individuelle Auswahl an digitalen Werbeträgern getroffen werden muss um eine möglichst zielgenau ausgesteuerte Werbekampagne zu erreichen.

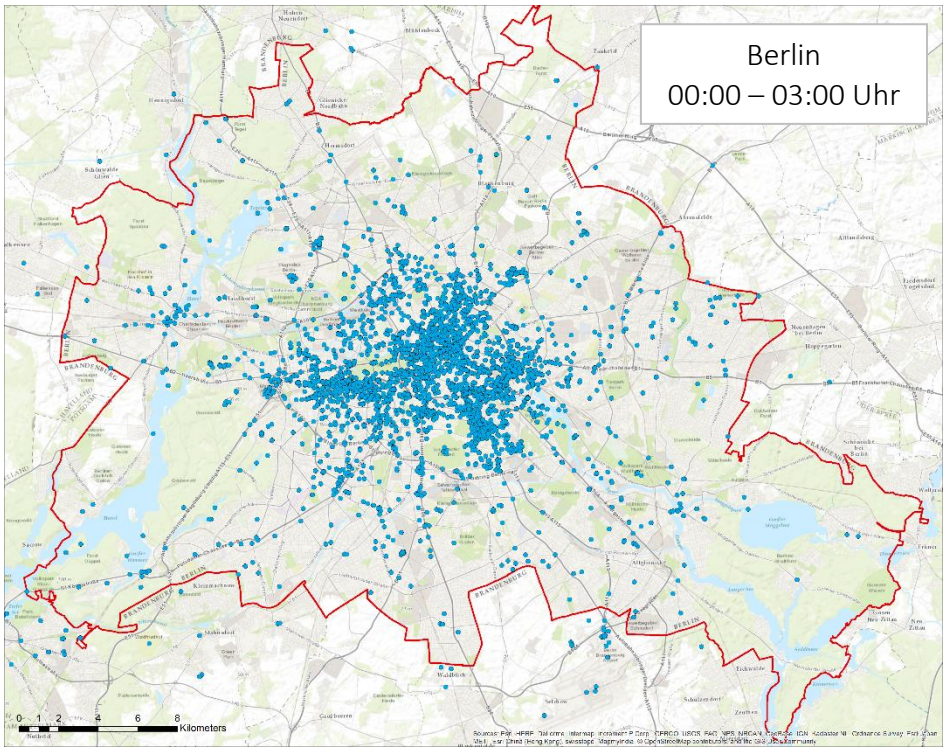


Abbildung 11: Tweets in der BoundingBox Berlin von 00:00 – 03:00 Uhr + Gemeindegrenze Berlin

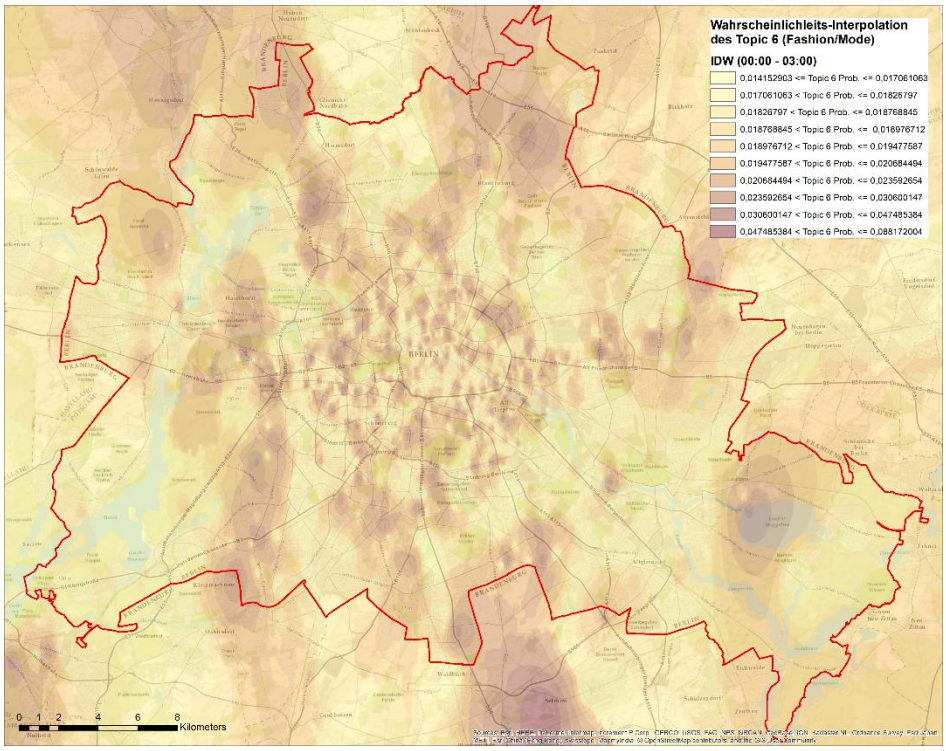


Abbildung 12: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 6 von 00:00 – 03:00 Uhr

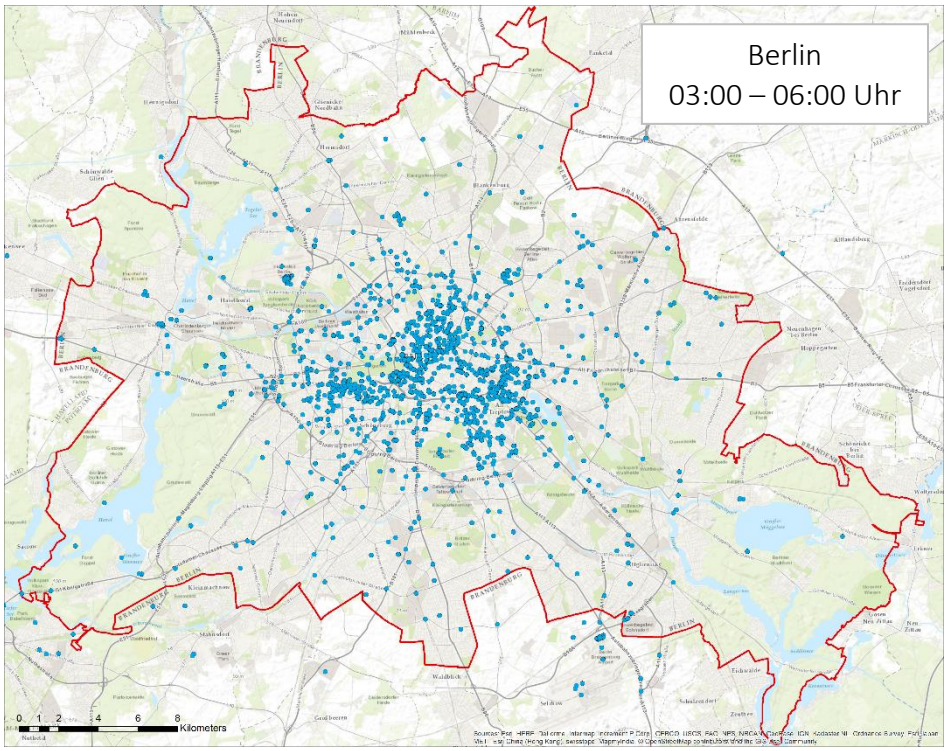


Abbildung 13: Tweets in der BoundingBox Berlin von 03:00 – 06:00 Uhr + Gemeindegrenze Berlin

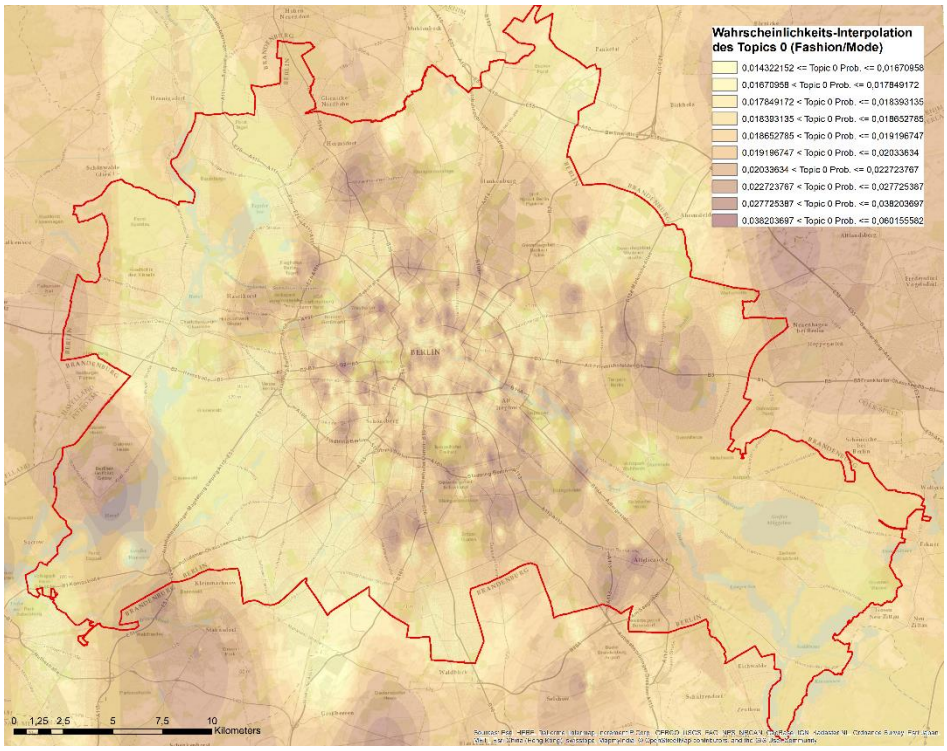


Abbildung 14: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 0 von 03:00 – 06:00 Uhr

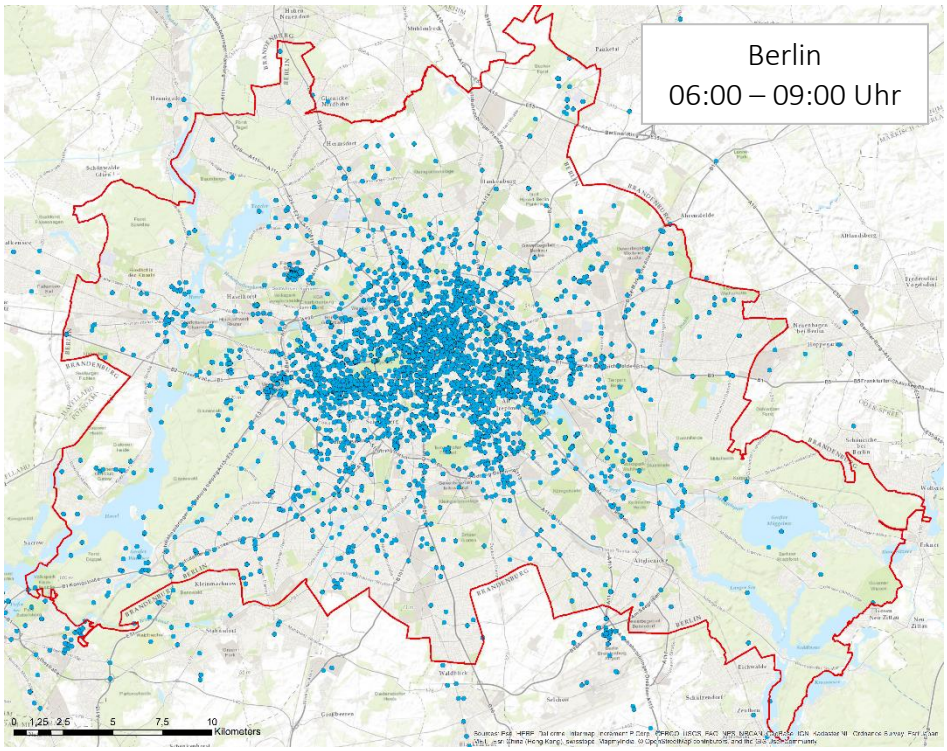


Abbildung 15: Tweets in der BoundingBox Berlin von 06:00 – 09:00 Uhr + Gemeindegrenze Berlin

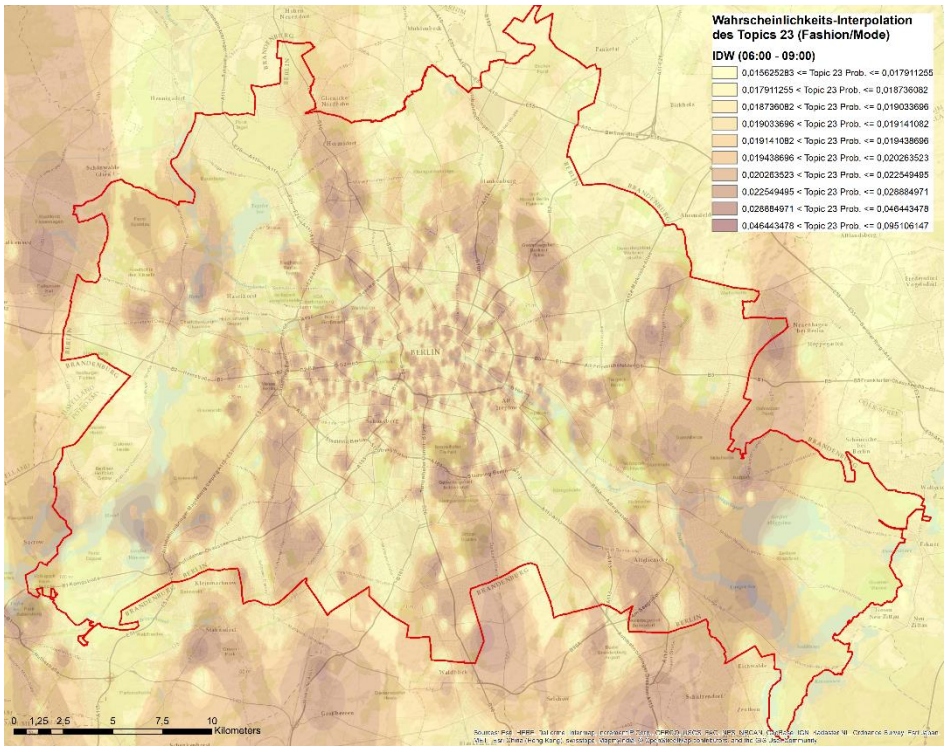


Abbildung 16: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 23 von 06:00 – 09:00 Uhr

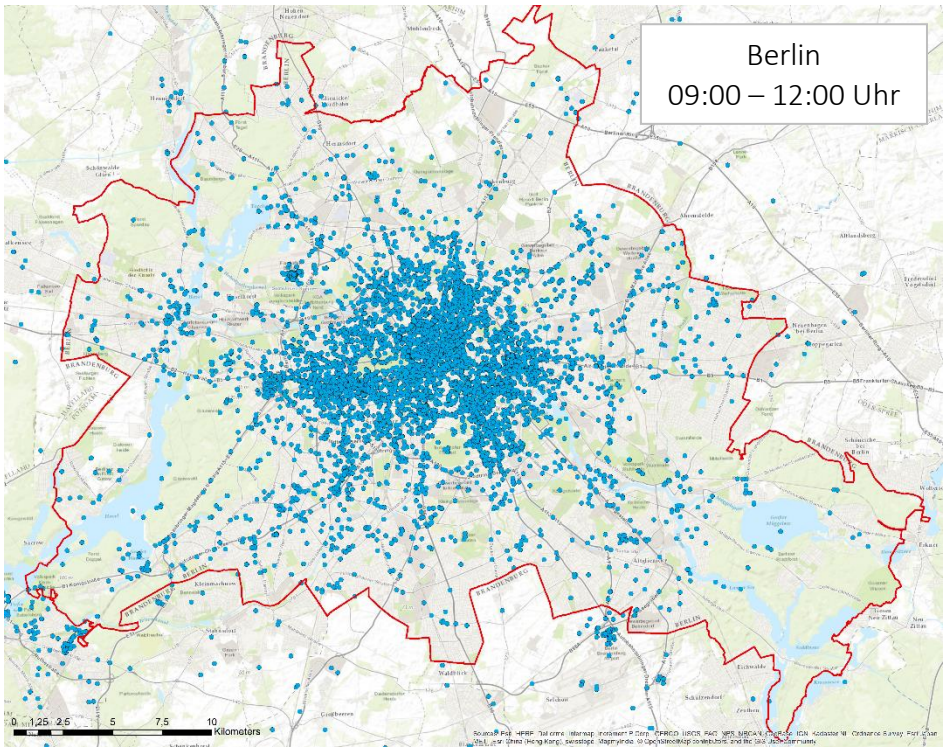


Abbildung 17: Tweets in der BoundingBox Berlin von 09:00 – 12:00 Uhr + Gemeindegrenze Berlin

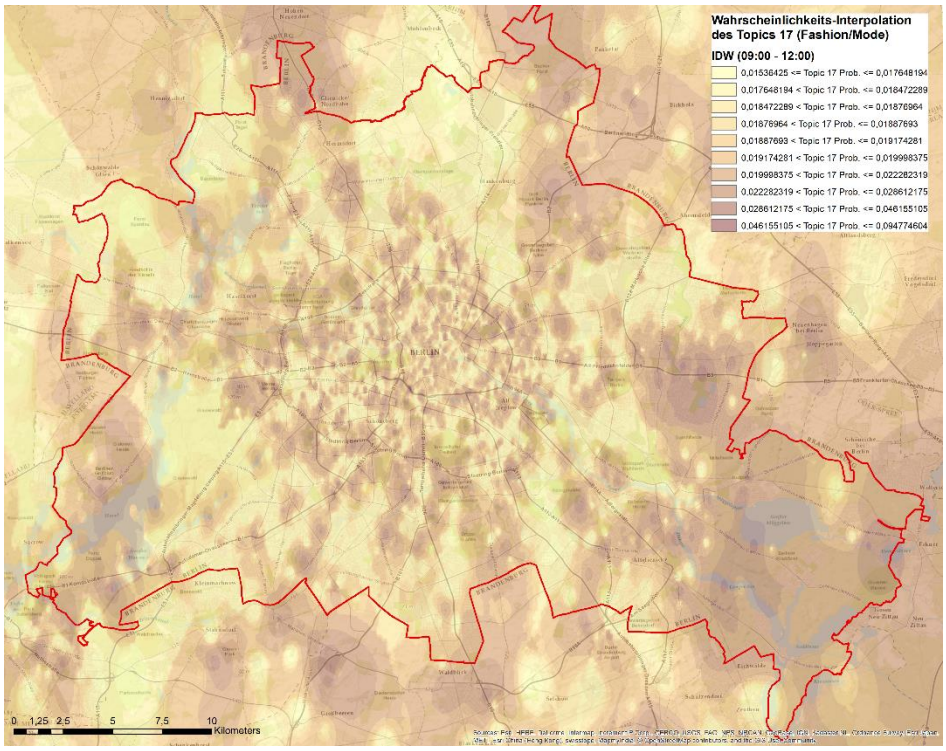


Abbildung 18: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 17 von 09:00 – 12:00 Uhr

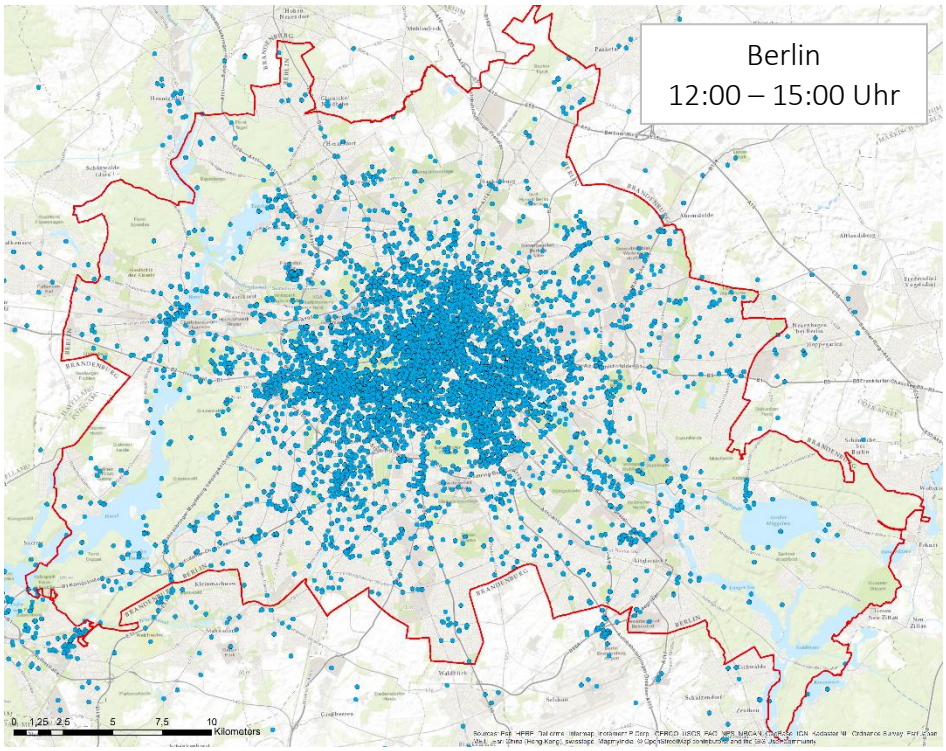


Abbildung 19: Tweets in der BoundingBox Berlin von 12:00 – 15:00 Uhr + Gemeindegrenze Berlin

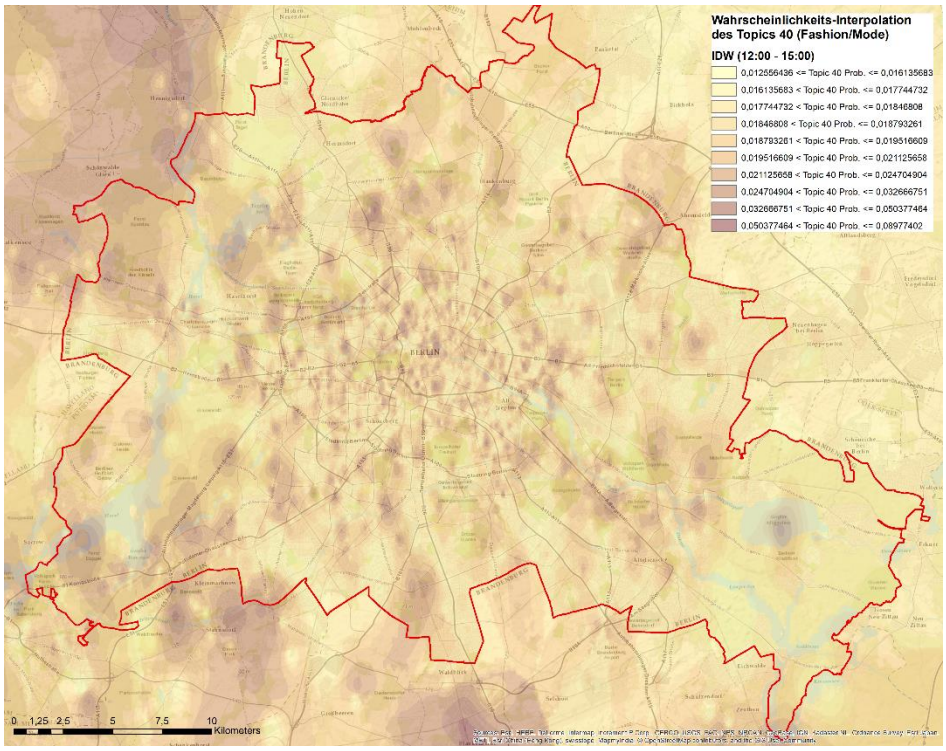


Abbildung 20: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 40 von 12:00 – 15:00 Uhr

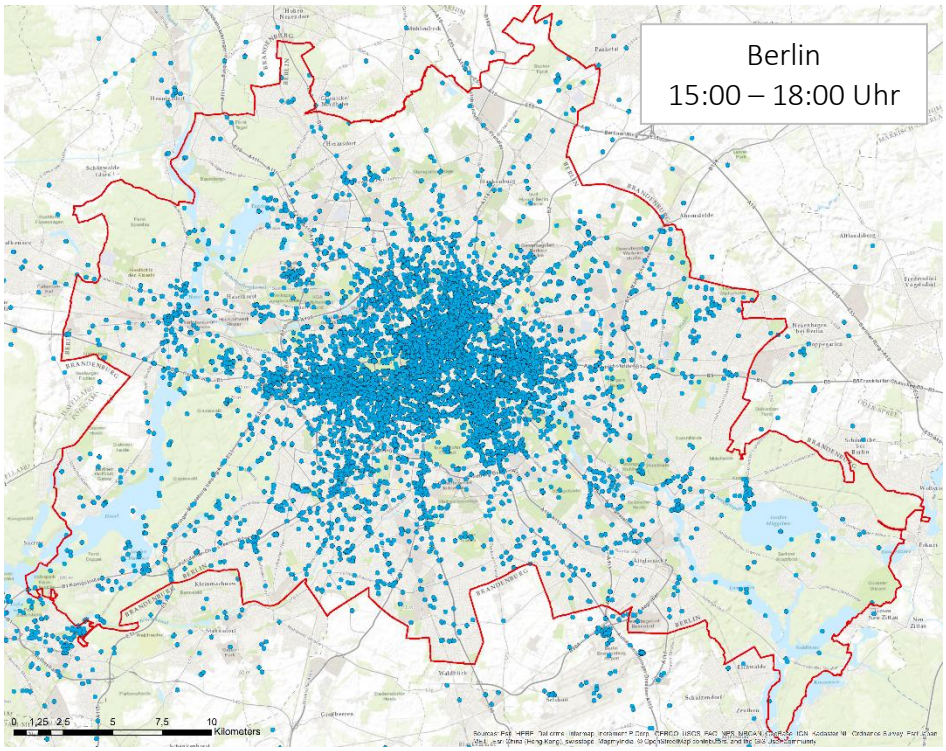


Abbildung 21: Tweets in der BoundingBox Berlin von 15:00 – 18:00 Uhr + Gemeindegrenze Berlin

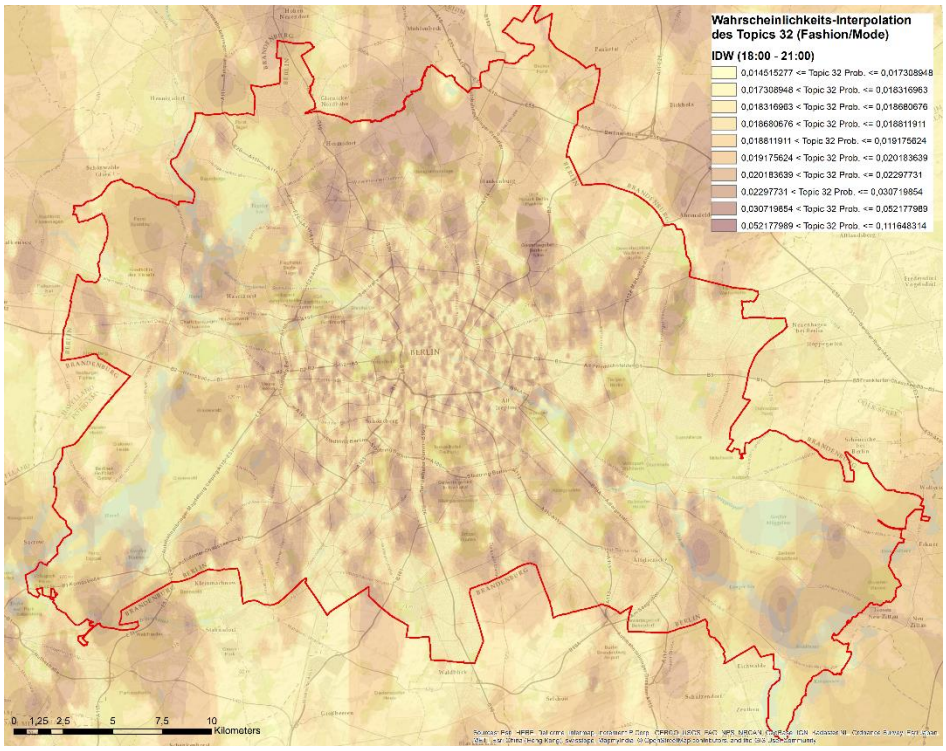


Abbildung 22: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 32 von 15:00 – 18:00 Uhr

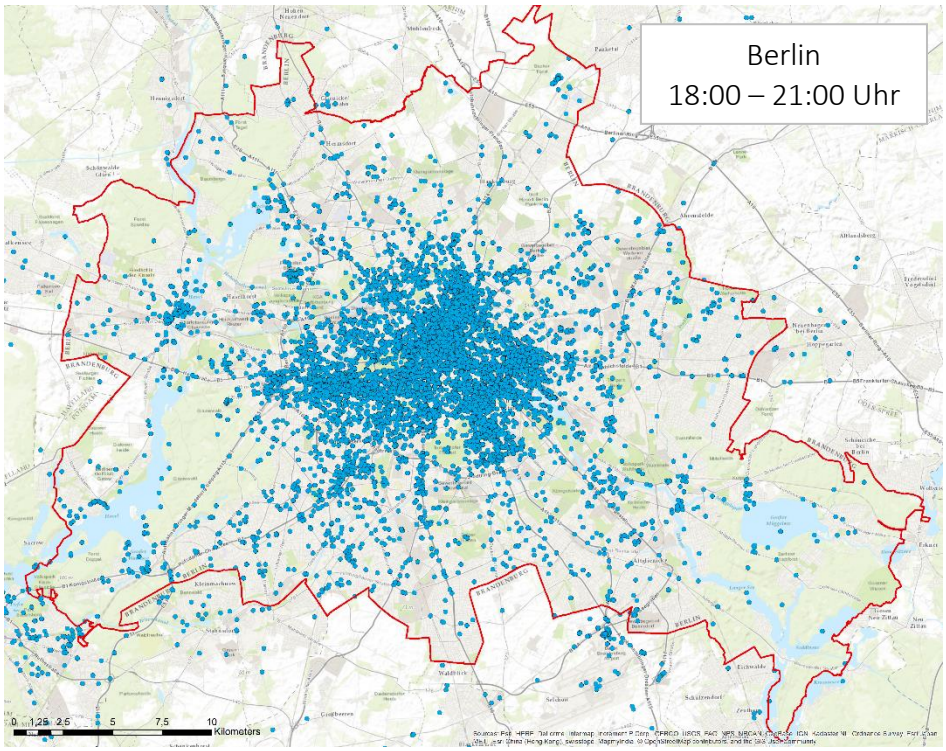


Abbildung 23: Tweets in der BoundingBox Berlin von 18:00 – 21:00 Uhr + Gemeindegrenze Berlin

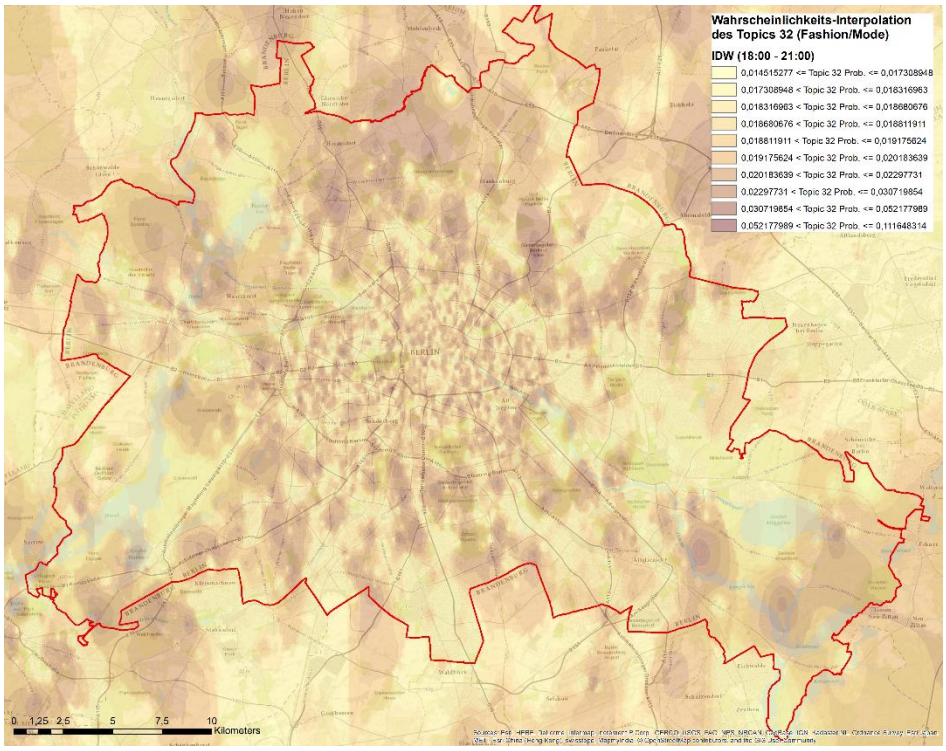


Abbildung 24: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 32 von 18:00 – 21:00 Uhr

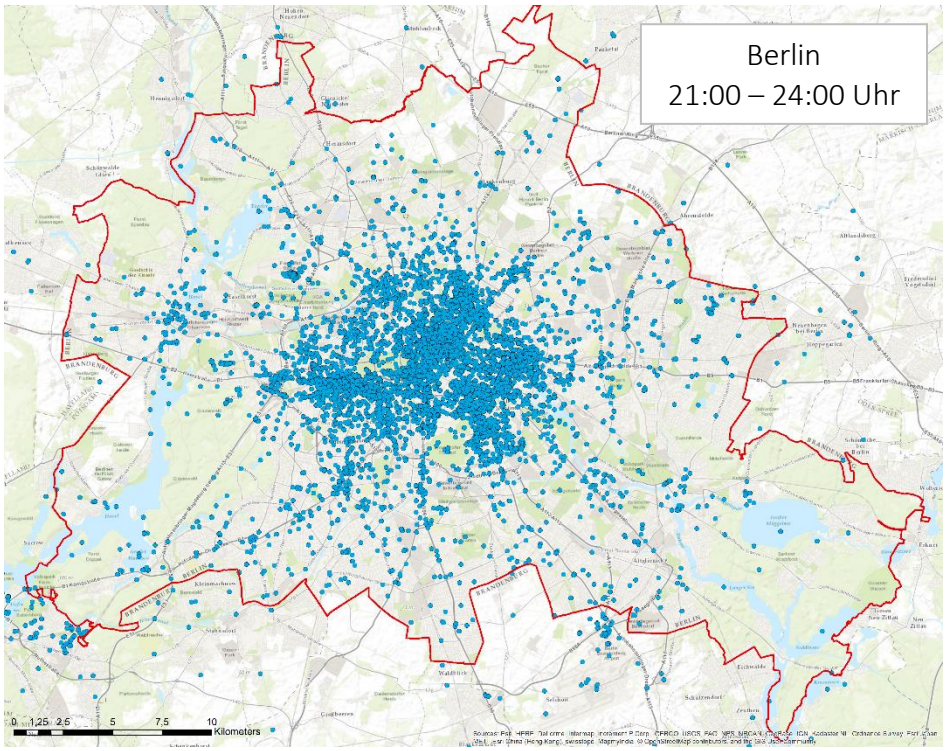


Abbildung 25: Tweets in der BoundingBox Berlin von 21:00 – 24:00 Uhr + Gemeindegrenze Berlin

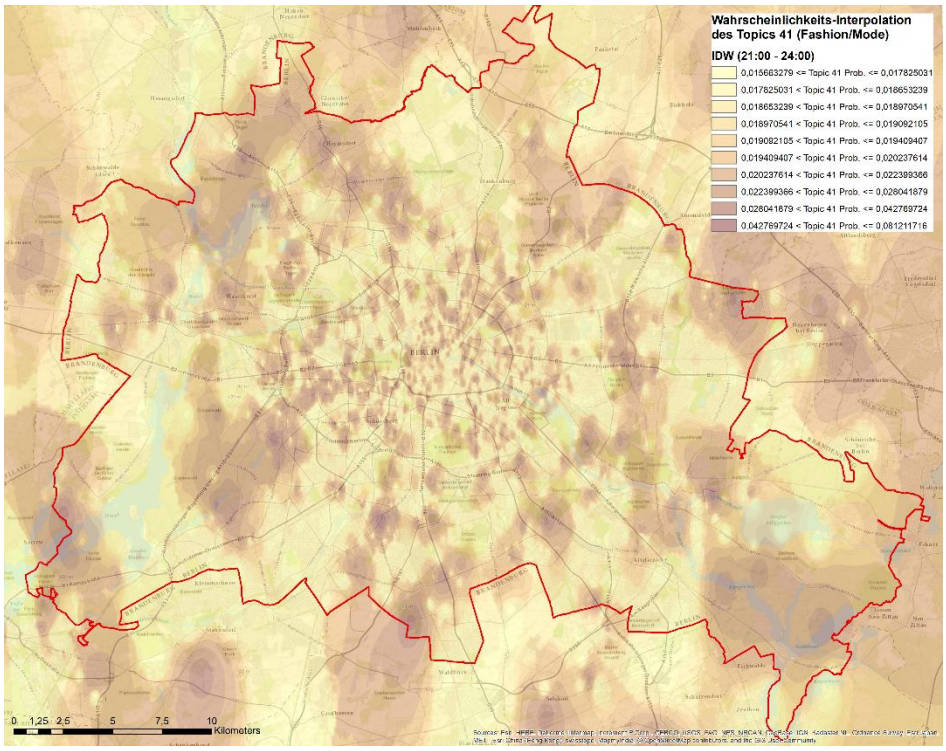


Abbildung 26: Interpolierte (IDW) Wahrscheinlichkeitsverteilung des Themas 41 von 21:00 – 24:00 Uhr

7.3.2 Standortempfehlung mit local Moran I

Im Folgenden wird für jede Zeitschiene eine mögliche Empfehlung an Standorten von digitalen Werbeträgern ermittelt. Die Standorte werden so gewählt, dass auf Basis der ermittelten Ergebnisse mit LDA, mit hoher Wahrscheinlichkeit Personen an oder in der Nähe der Standorte sind, die ein erhöhtes Interesse an Fashion/Mode aufweisen.

Um die erhobenen Punkte der Textbeiträge/Tweets räumlich zu clustern, wird mit der räumlichen Autokorrelation gearbeitet. Auf Basis der Koordinaten und der jeweiligen Themen-Wahrscheinlichkeit für Fashion/Mode wird der Anselins local Moran I (siehe Kapitel *7.3.2.1 Local Morans I*) ermittelt. Zusammen mit dem ermittelten Wert I kann der Zi-Score errechnet werden. Dieser erlaubt es, die Daten in unterschiedliche Cluster einzuteilen. Werte mit einem hohen Zi-Score lassen sich unter dem räumlichen Autokorrelations-Cluster *HH* zusammenfassen. Alle Punkte der Textbeiträge, welche unter das Cluster *HH* fallen, werden extrahiert und mit einem Buffer von 500m versehen. Anschließend werden die Standorte von digitalen Werbeträgern ermittelt, welche sich sowohl in diesem Buffer-Gebiet befinden, als auch in den Stadtgrenzen von Berlin. Abbildung 27 zeigt die Aufbereitungsprozesse welche mit Hilfe der Software ArcMap durchgeführt werden.

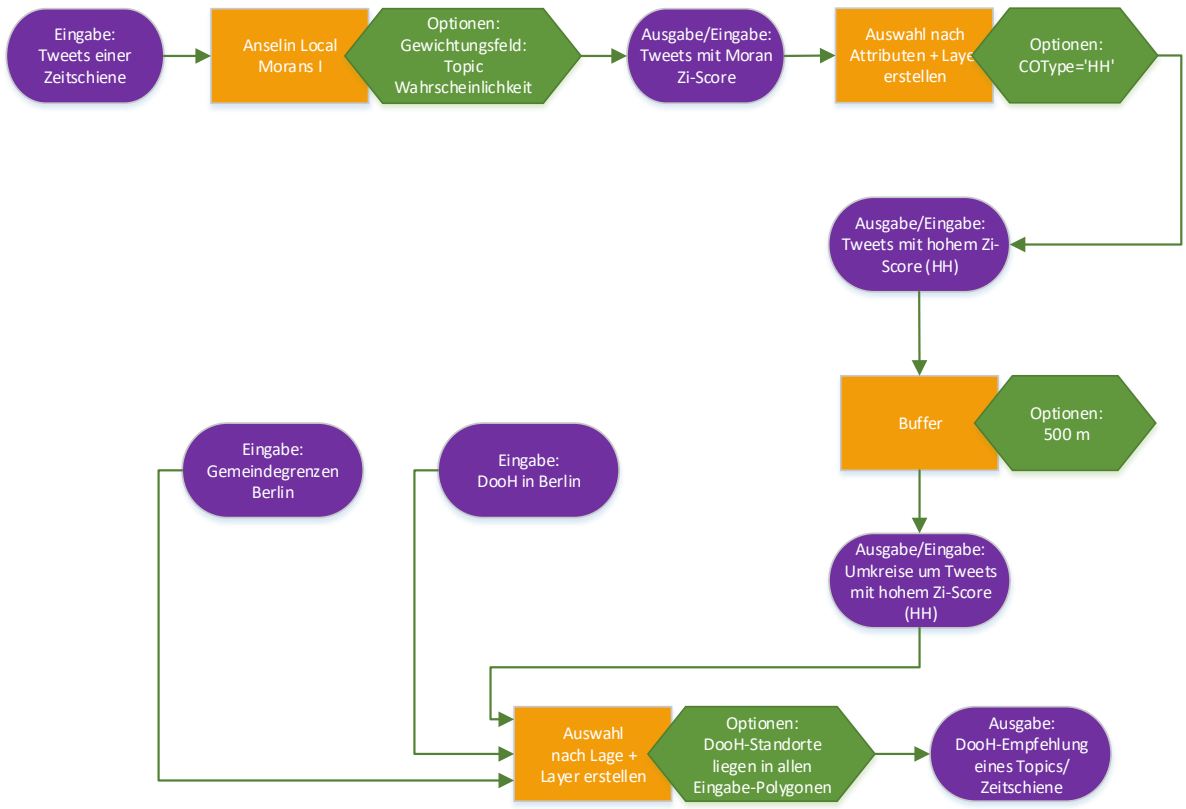


Abbildung 27: Abläufe in ArcMap zur Ermittlung einer DooH-Mediaempfehlung basierend auf dem Moran Zi-Score der Themen-Wahrscheinlichkeiten eines Themas und einer Zeitschiene

7.3.2.1 Local Morans I

Der local Morans I ist ein Maß für die räumliche Autokorrelation. Diese wird auf Basis der räumlichen Position der betrachteten Geometrien und deren definierten Attributs-Werten ermittelt. Dabei wird der Attributs-Wert einer jeden Geometrie mit allen anderen Geometrien verglichen. Die Beziehung zwischen jedem betrachteten Geometrie-Paar wird gewichtet, wobei Geometrien die näher beieinander liegen höher gewichtet werden als Geometrien, die weiter auseinander liegen (*Anselin, 1995*). Die Formeln (1) und (2) geben an, wie der Wert local Moran I berechnet wird.

$$I_i = \frac{x_i - \bar{X}}{S_i^2} \sum_{j=1, j \neq i}^{n} \omega_{i,j}(x_i - \bar{X}) \quad (1)$$

I_i=local Moran I

x_i=Attribut einer Geometrie an Position i

$\bar{X}$=Arithmetisches Mittel der entsprechenden Attribute

$\omega_{i,j}$=räumliche Gewichtung zwischen den Geometrien i und j

$$S_i^2 = \frac{\sum_{j=1, j \neq i}^{n} (x_i - \bar{X})^2}{\mathrm{n} - 1} \quad (2)$$

n=Anzahl aller betrachteten Geometrien

Der ermittelte Wert I_i liegt immer zwischen -1 und +1. Je höher der Wert ist, umso räumlich gleichartiger ist die Geometrie im Vergleich zu deren umliegenden Geometrien. Je niedriger der Wert ist, umso räumlich unähnlicher ist die Geometrie im Vergleich zu deren umliegenden Geometrien.

Mit dem local Moran I lässt sich die statistische Signifikanz ermitteln. Dabei wird jeder errechnete Wert I_i mit dem theoretisch zu erwartenden Werten verglichen. Die Formeln (3) – (5) geben an, wie der z-score z_{I_i} berechnet werden kann.

$$E[I_i] = -\frac{\sum_{j=1, j \neq i}^{n} \omega_{ij}}{\mathrm{n} - 1} \quad (3)$$

$$V[I_i] = E[I_i^2] - E[I_i]^2 \quad (4)$$

$$z_{I_i} = \frac{I_i - E[I_i]}{\sqrt{V[I_i]}} \quad (5)$$

Ein hoher positiver Wert für z_{I_i} sagt aus, dass die umliegenden Geometrie-Werte ähnlich hohe oder ähnlich niedrige Werte aufweisen. Demnach lassen hier zwei Cluster ableiten: Alle z_{I_i} mit einem hohen Wert bilden den räumlichen Autokorrelations-Cluster *HH.* Alle z_{I_i} mit einem niedrigen Wert bilden den räumliche Autokorrelations-Cluster *LL (Anselin, 1993).*

7.3.2.2 Ergebnisse

Die Abbildung 28 - Abbildung 43 zeigen pro Zeitschiene die Ergebnisse, die mit den Prozessen aus Abbildung 27 gewonnen wurden. Die Prozesse wurden je Zeitschiene und dem dazu identifiziertem Thema für Fashion/Mode durchgeführt. Dabei zeigt die jeweils erste Abbildung die Punkte, die durch die Clusterung der räumlichen Autokorrelation (local Moran I) ermittelt worden sind. Dem folgt je Zeitschiene eine Abbildung mit einem Buffer von 500 Meter um die ermittelten Punkte sowie die Standorte von digitalen Werbeträgern, die in diesem Buffer und den Stadtgrenzen von Berlin liegen.

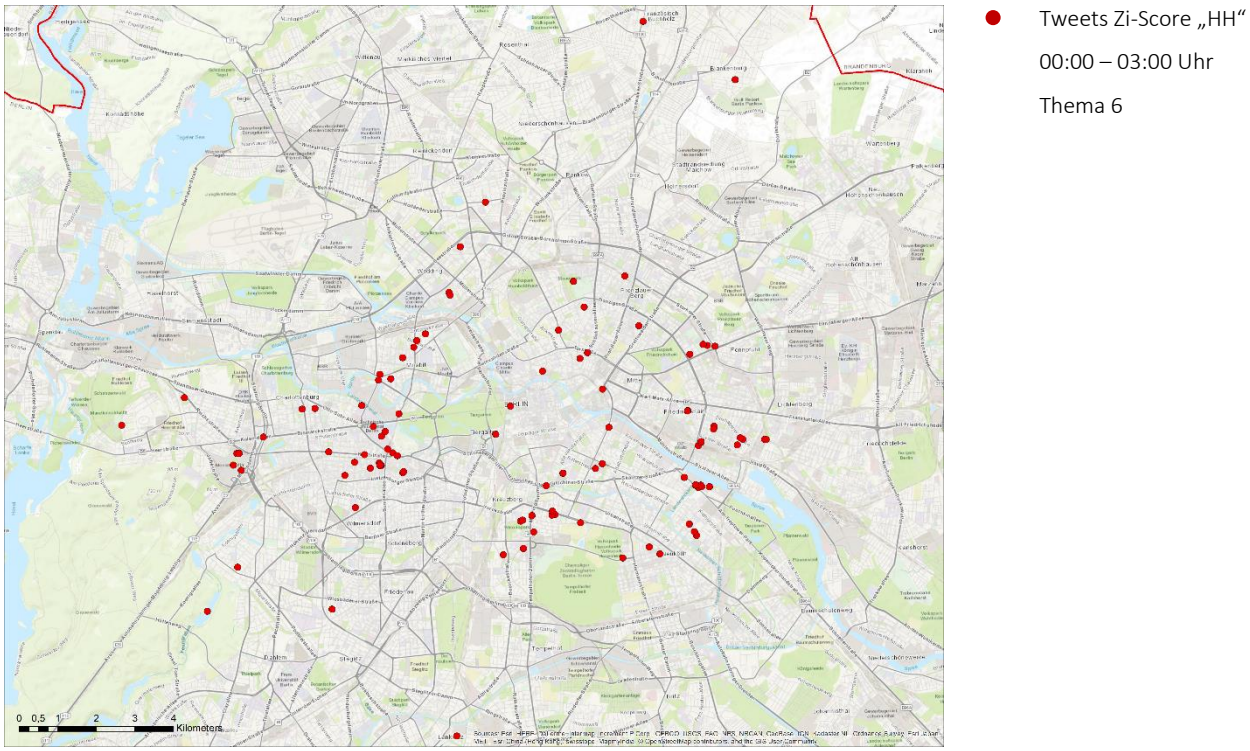


Abbildung 28: Tweets von 00:00 – 03:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 6 in Berlin

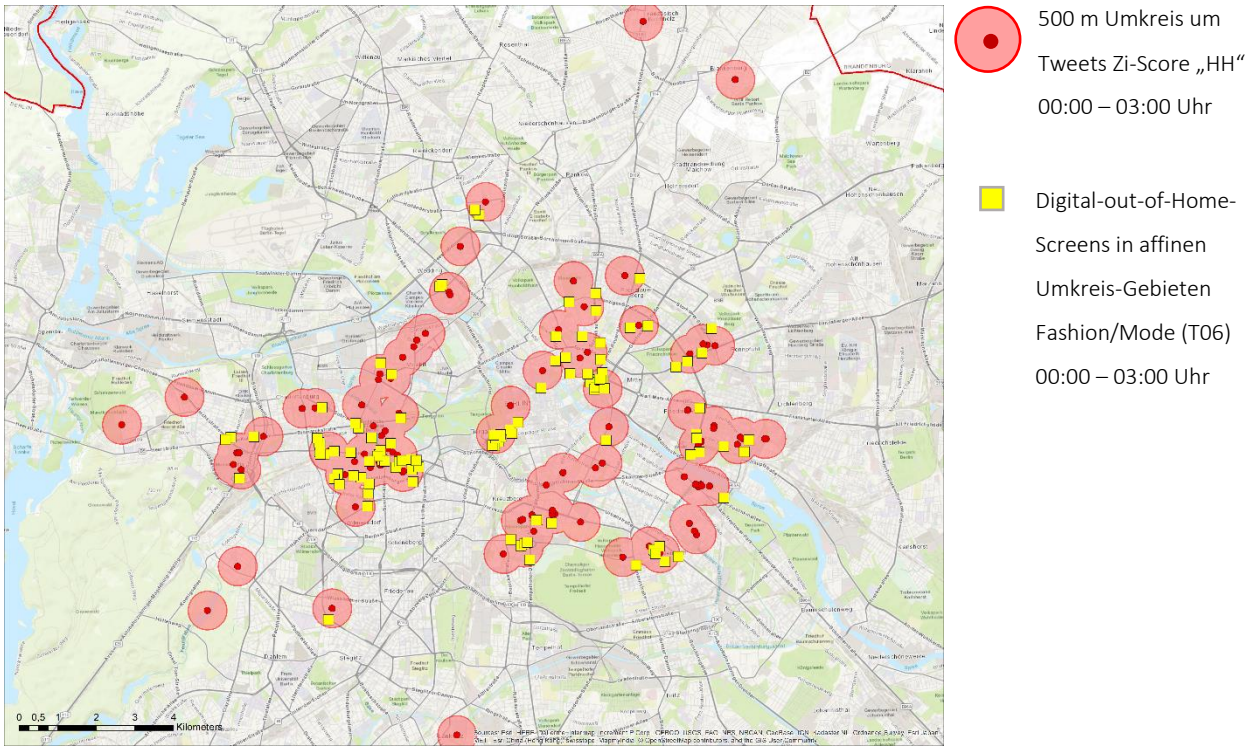


Abbildung 29: DooH-Empfehlung für Fashion/Mode-Interessierte von 00:00 – 03:00 Uhr

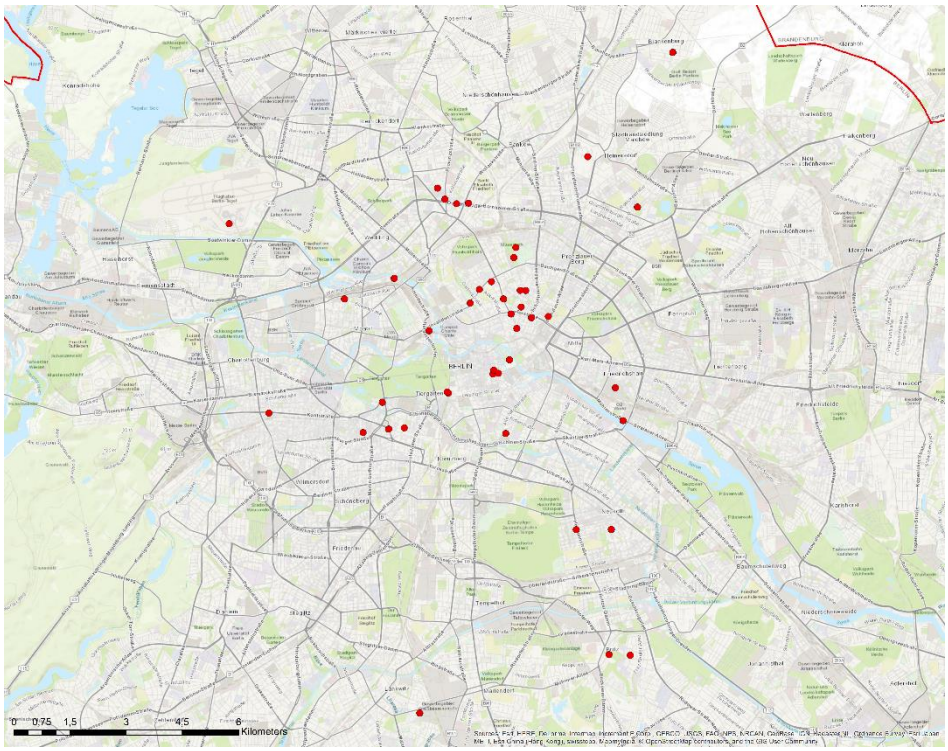


Abbildung 30: Tweets von 03:00 – 06:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 0 in Berlin

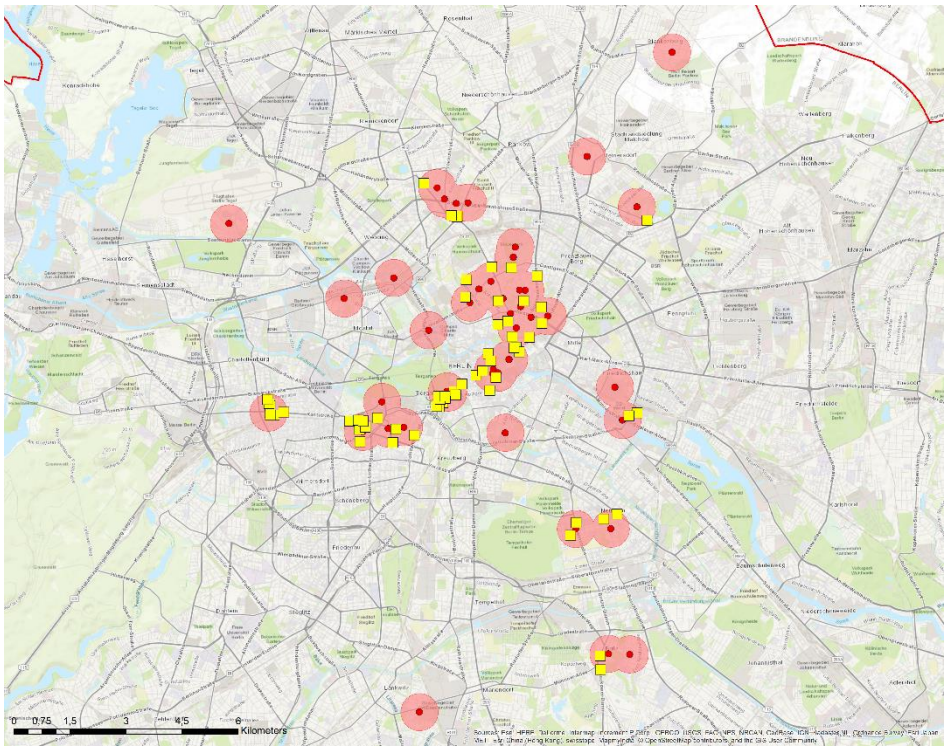


Abbildung 31: DooH-Empfehlung für Fashion/Mode-Interessierte von 03:00 – 06:00 Uhr

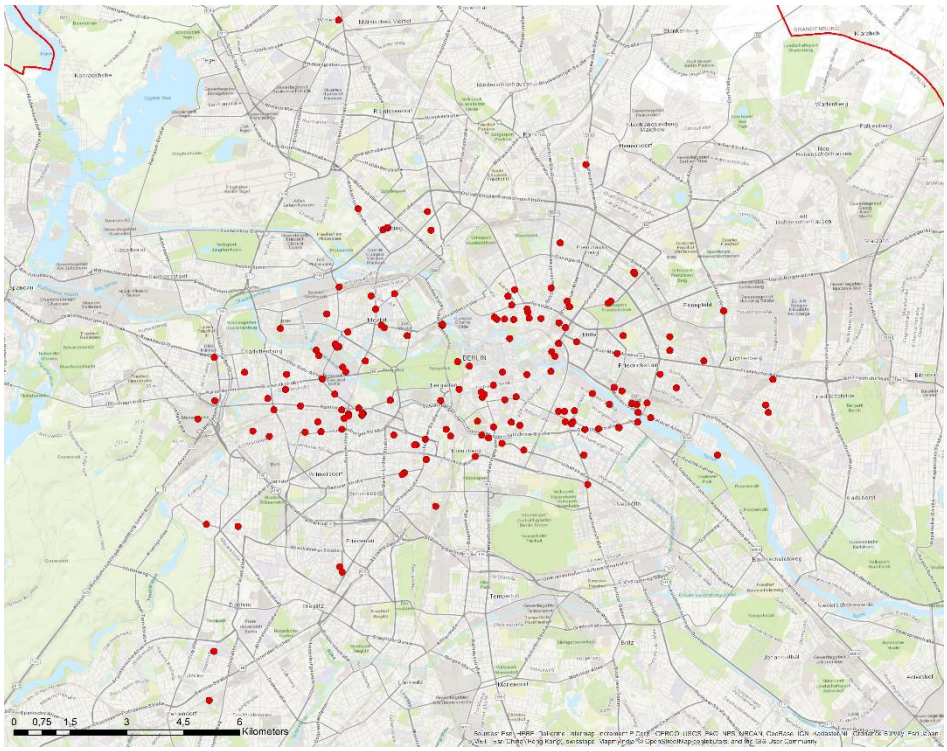


Abbildung 32: Tweets von 06:00 – 09:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 23 in Berlin

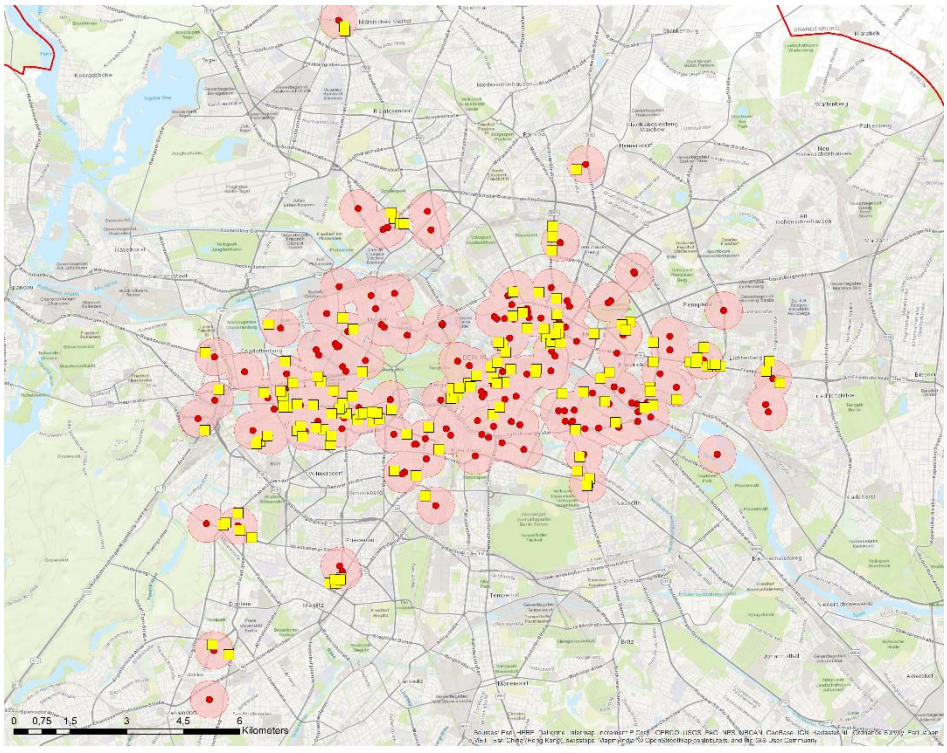
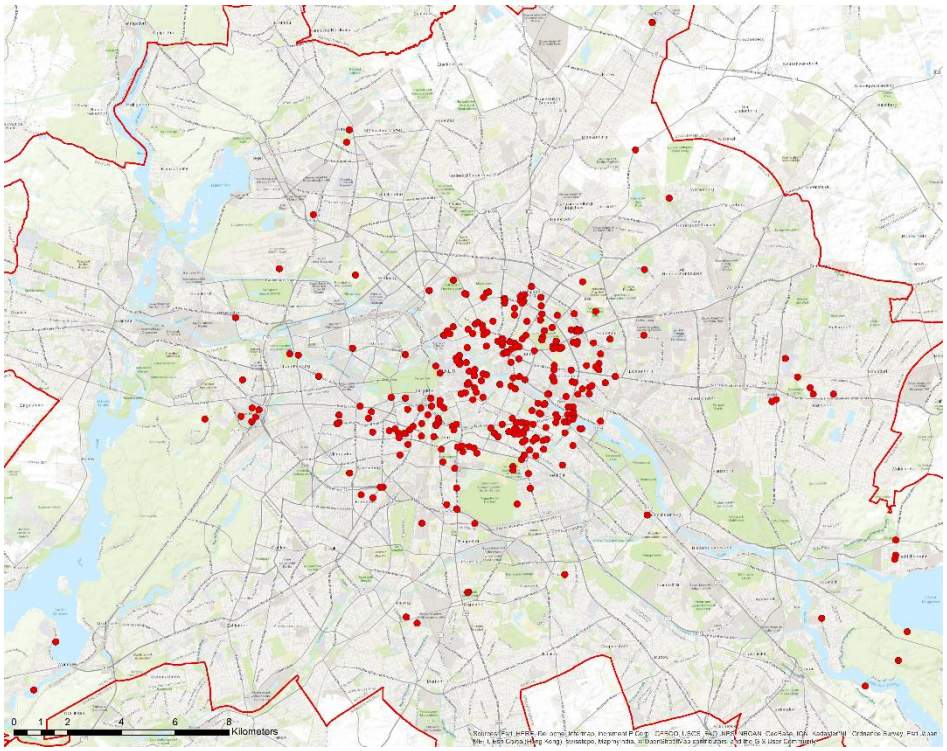
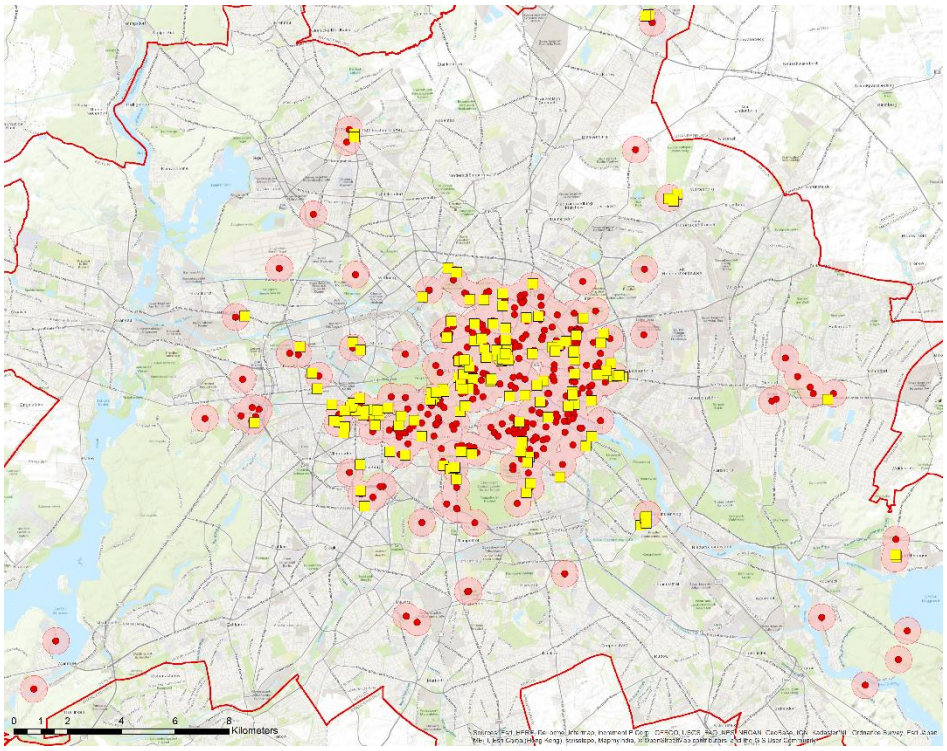


Abbildung 33: DooH-Empfehlung für Fashion/Mode-Interessierte von 06:00 – 09:00 Uhr



Tweets Zi-Score „HH“
09:00 – 12:00 Uhr
Thema 17

Abbildung 34: Tweets von 09:00 – 12:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 17 in Berlin



500 m Umkreis um
Tweets Zi-Score „HH“
06:00 – 09:00 Uhr

Digital-out-of-Home-Screens in affinen Umkreis-Gebieten Fashion/Mode (T17)
09:00 – 12:00 Uhr

Abbildung 35: DooH-Empfehlung für Fashion/Mode-Interessierte von 09:00 – 12:00 Uhr

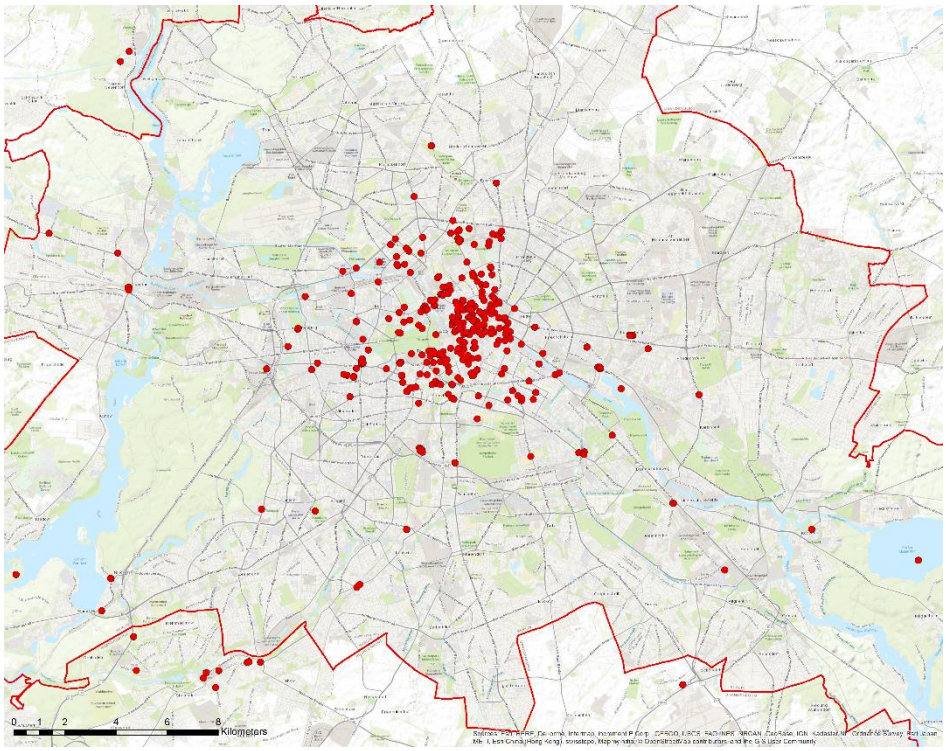


Abbildung 36: Tweets von 12:00 – 15:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 40 in Berlin

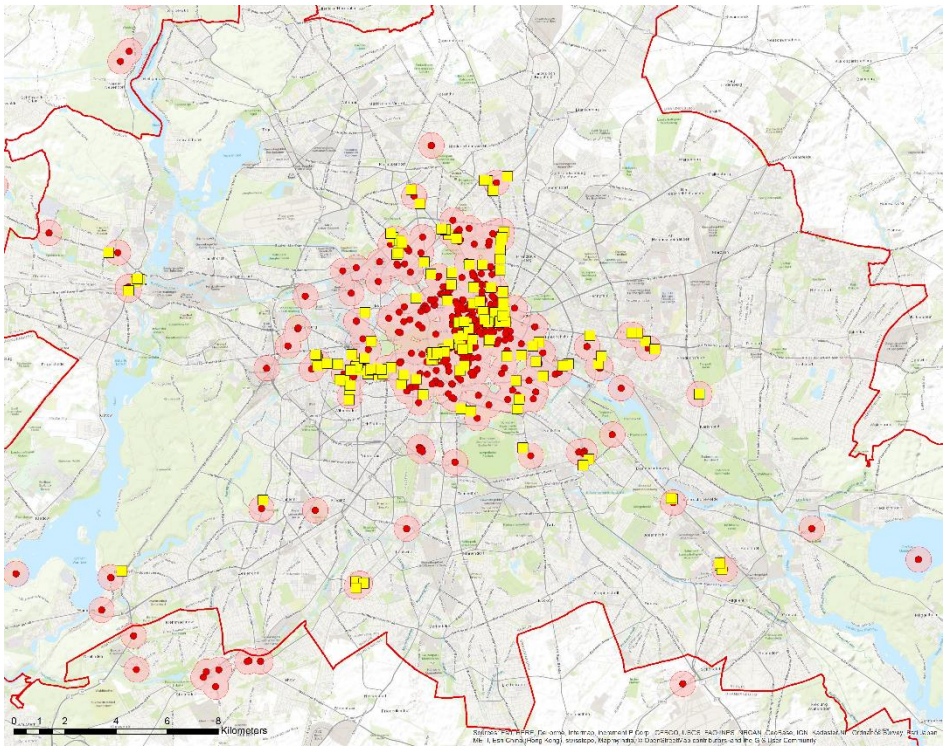


Abbildung 37: DooH-Empfehlung für Fashion/Mode-Interessierte von 12:00 – 15:00 Uhr

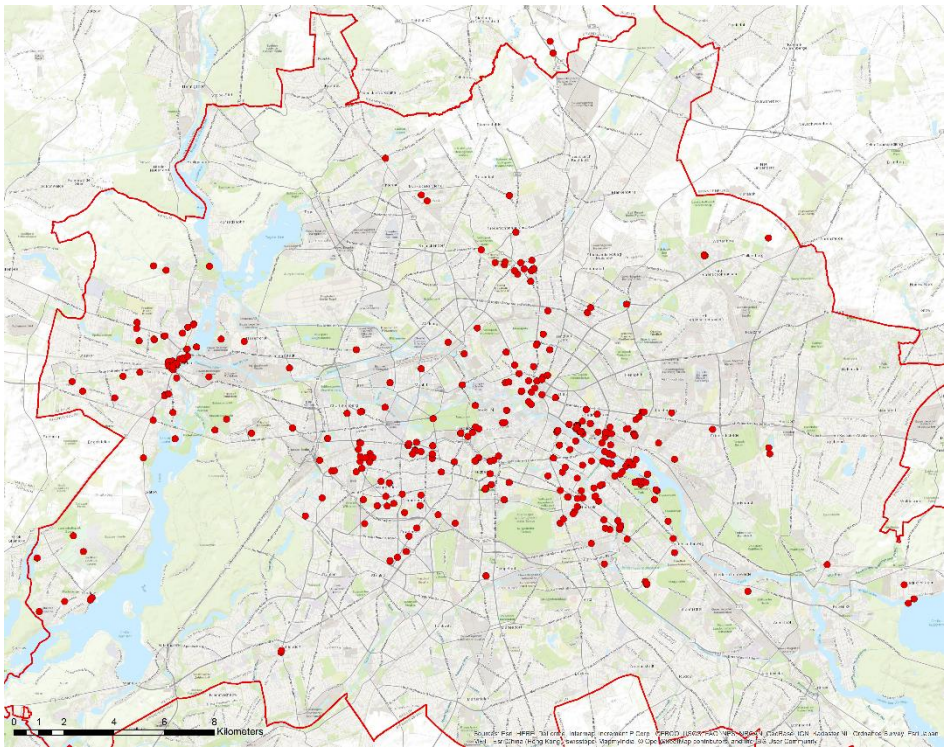


Abbildung 38: Tweets von 15:00 – 18:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 40 in Berlin

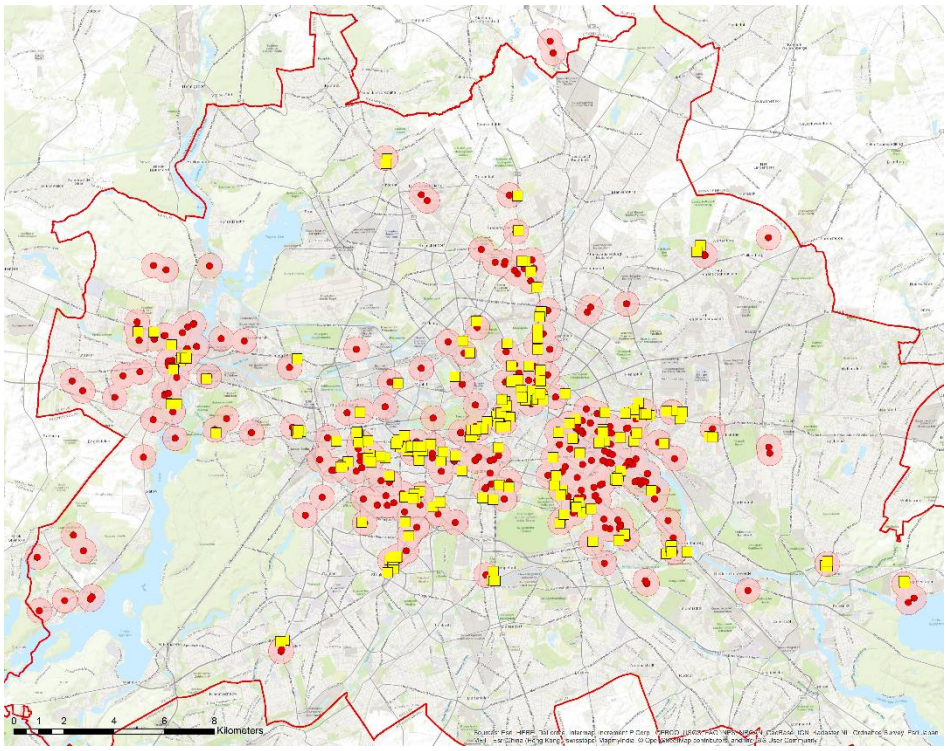


Abbildung 39: DooH-Empfehlung für Fashion/Mode-Interessierte von 15:00 – 18:00 Uhr

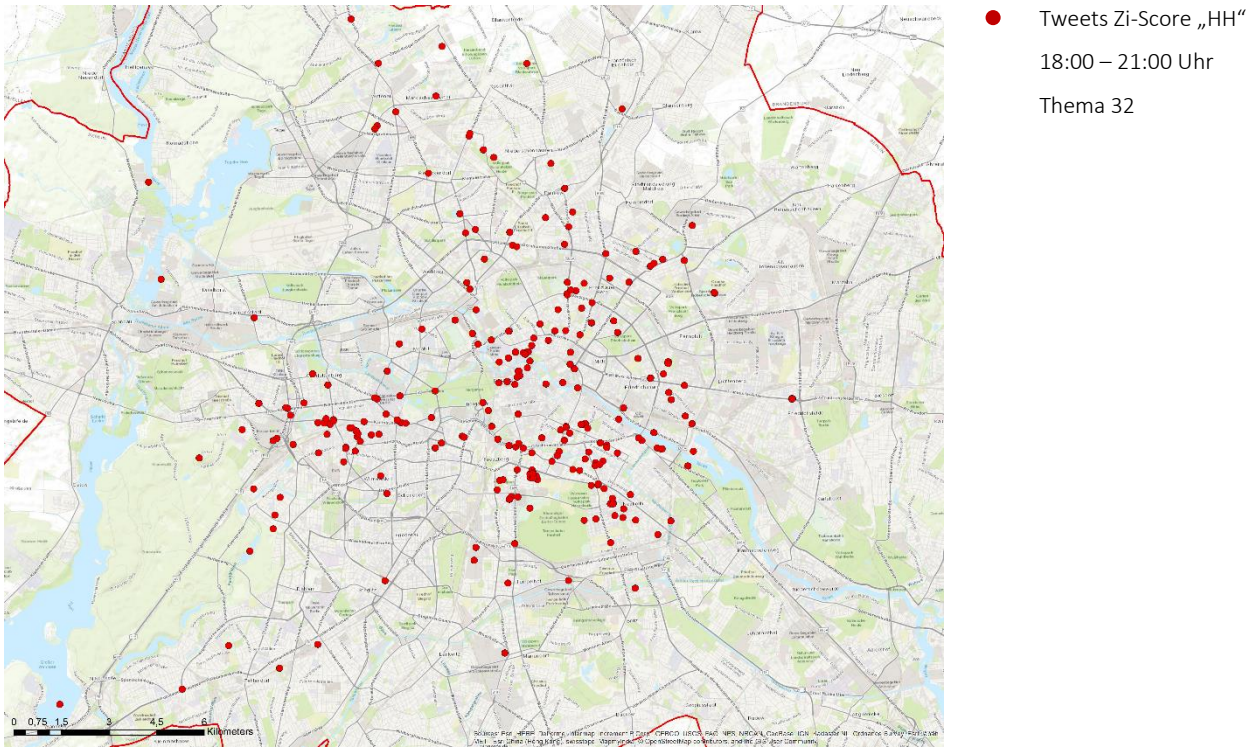


Abbildung 40: Tweets von 18:00 – 21:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 32 in Berlin

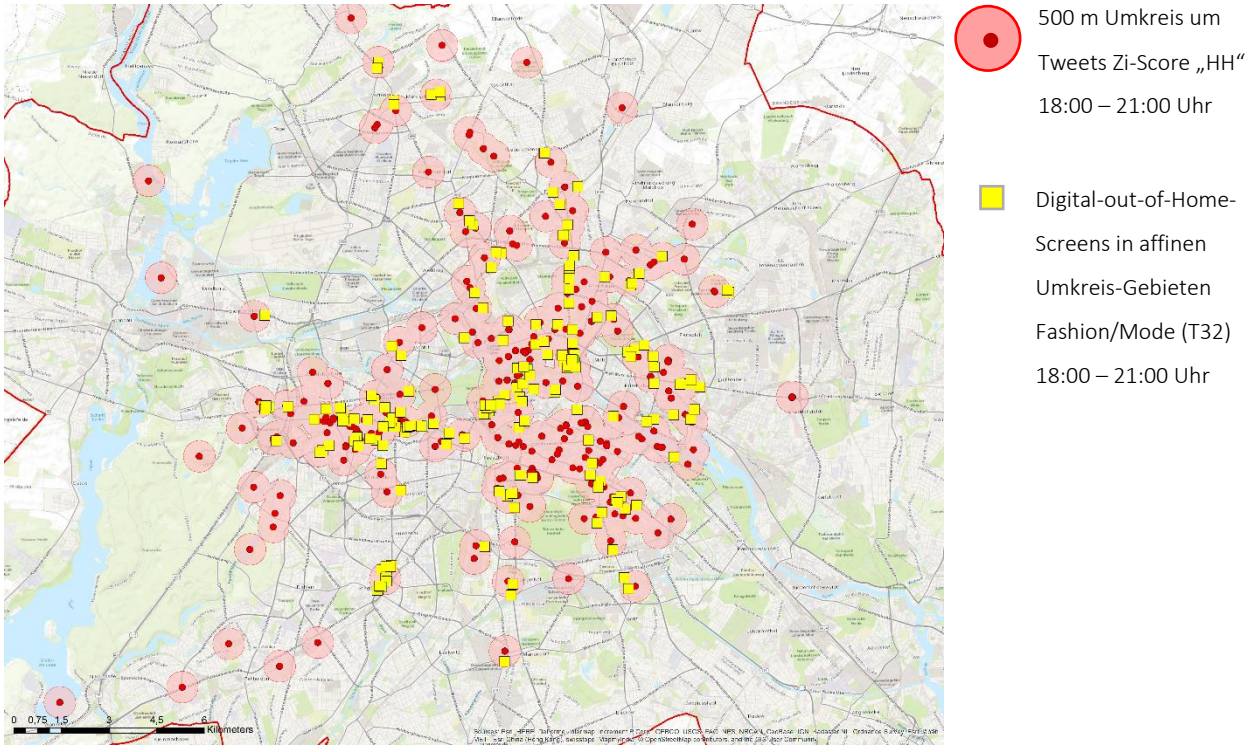


Abbildung 41: DooH-Empfehlung für Fashion/Mode-Interessierte von 18:00 – 21:00 Uhr

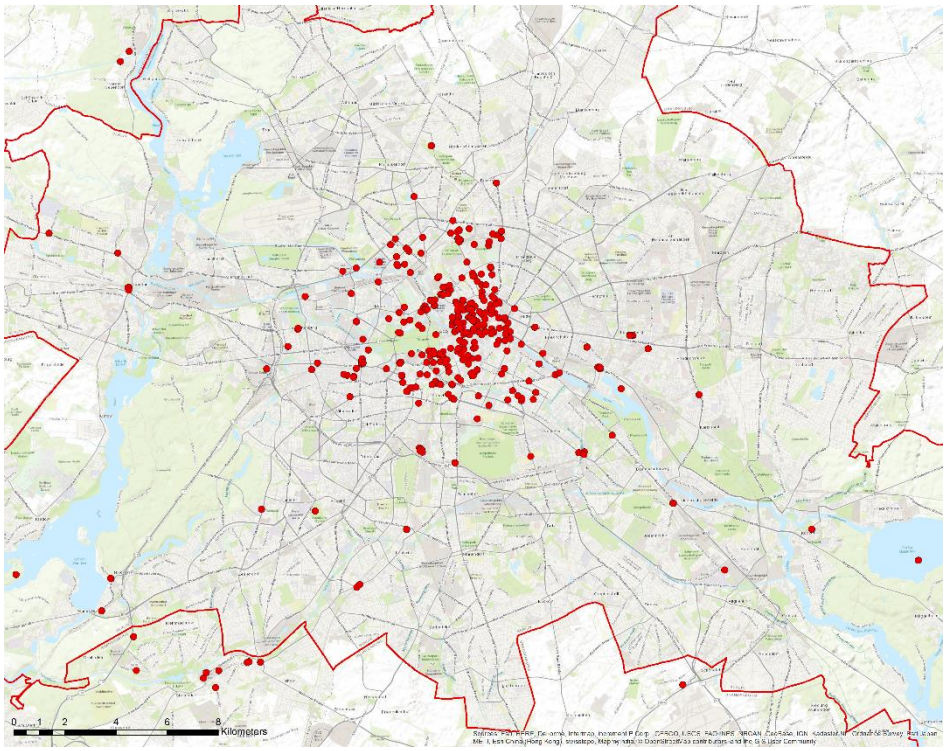


Abbildung 42: Tweets von 21:00 – 24:00 Uhr mit Moran Zi-Score Cluster „HH“ des Themas 41 in Berlin

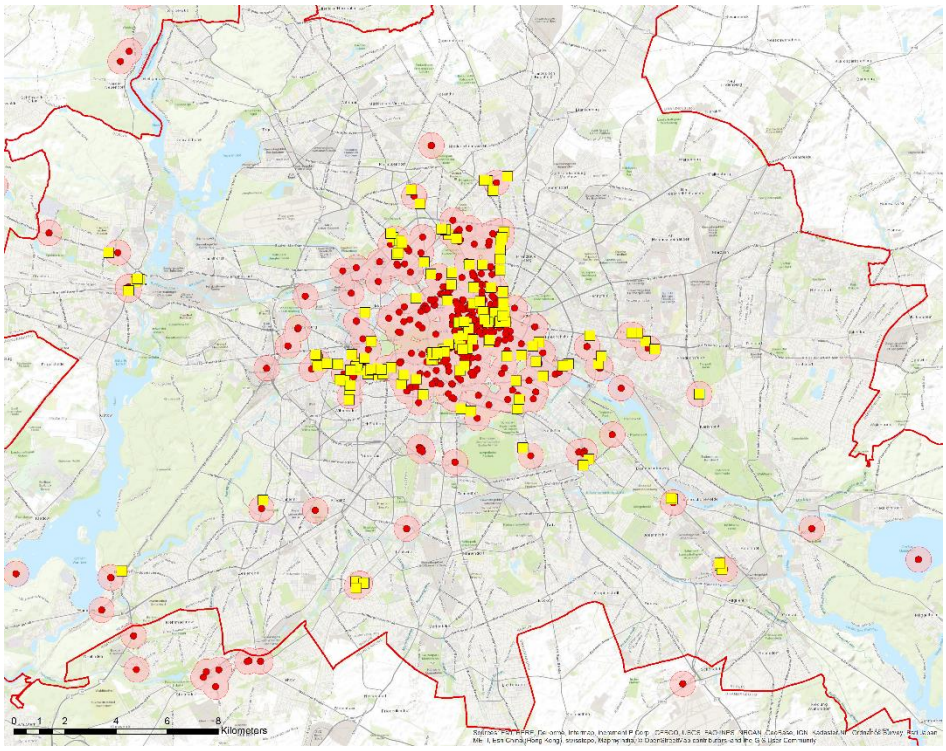


Abbildung 43: DooH-Empfehlung für Fashion/Mode-Interessierte von 21:00 – 24:00 Uhr

Anbieter von digitalen Werbeträgern haben Ihre Screens i.d.R. in unterschiedliche Netze aufgeteilt, welche Rückschlüsse über die Standorte der Screens zulassen. Alle Screens, welche z.B. dem Netz „*McDonalds-TV*“ angehören, sind in den Fast-Food-Filialen von McDonalds oder in deren Außenbereichen positioniert. Die Tabelle 5 - Tabelle 7 geben an, welchem Netz die ermittelten Standorte von digitalen Werbeträgern angehören und die Anzahl der Standorte des jeweiligen Netzes.

Tabelle 5: Anzahl Standorte in digitalen-Werbeträger-Netzen der ermittelten DooH-Empfehlung 00:00 - 09:00 Uhr

00:00 - 03:00		03:00 - 06:00		06:00 - 09:00	
Netze	**Anzahl**	**Netze**	**Anzahl**	**Netze**	**Anzahl**
TV-Wartezimmer	41	TV-Wartezimmer	22	TV-Wartezimmer	52
Gastro Channel	14	Campus-TV	11	Gastro Channel	21
Campus-TV	9	Gastro Channel	9	Campus-TV	15
adpack	8	adpack	6	adpack	13
Active-TV	7	Active-TV	5	UNICUM TV	11
Hairstyling-TV	7	EDEKA TV	5	Active-TV	10
Kaisers TV	7	Hairstyling-TV	5	EDEKA TV	9
UNICUM TV	7	Apotheken-TV	3	Hairstyling-TV	8
EDEKA TV	6	Cash-TV	3	Cash-TV	7
Cash-TV	5	Intersport-TV	3	Kaisers TV	7
King Channel	5	King Channel	3	McDonalds-TV	7
McDonalds-TV	5	McDonalds-TV	3	Cinema Channel	6
Cinema Channel	4	UNICUM TV	3	Apotheken-TV	4
Intersport-TV	4	Kaisers TV	2	Intersport-TV	4
Apotheken-TV	3	Saturn-TV	2	Saturn-TV	4
Media Markt-TV	3	Stage-TV	2	Airport Berlin	3
Saturn-TV	3	Cinema Channel	1	King Channel	3
Stage-TV	3	Cinemaxx	1	Stage-TV	3
Kaufland TV	2	Hochschulwerbung	1	Cinema-TV	2
Cinema-TV	1	Media Markt-TV	1	Media Markt-TV	2
Cinemaxx	1	MEDIMAX-TV	1	MEDIMAX-TV	2
DCLP	1			Cinemaxx	1
MEDIMAX-TV	1			DCLP	1
				Hochschulwerbung	1

Tabelle 6: Anzahl Standorte in digitalen-Werbeträger-Netzen der ermittelten DooH-Empfehlung 09:00 - 18:00 Uhr

09:00 - 12:00		12:00 - 15:00		15:00 - 18:00	
Netze	**Anzahl**	**Netze**	**Anzahl**	**Netze**	**Anzahl**
TV-Wartezimmer	56	TV-Wartezimmer	53	TV-Wartezimmer	81
Gastro Channel	25	Campus-TV	24	Gastro Channel	22
adpack	14	Gastro Channel	18	adpack	19
EDEKA TV	12	UNICUM TV	14	Active-TV	18
Active-TV	9	adpack	12	EDEKA TV	16
Campus-TV	9	EDEKA TV	12	Apotheken-TV	13
Kaisers TV	9	Active-TV	11	Hairstyling-TV	13
Cash-TV	8	Cash-TV	11	Campus-TV	12
Hairstyling-TV	8	Hairstyling-TV	10	Cash-TV	11
McDonalds-TV	7	McDonalds-TV	9	Kaisers TV	10
UNICUM TV	7	Kaisers TV	6	McDonalds-TV	9
Apotheken-TV	6	King Channel	6	King Channel	8
Saturn-TV	6	Saturn-TV	6	Saturn-TV	8
King Channel	5	Apotheken-TV	5	Cinema Channel	7
Cinema Channel	4	Cinema Channel	5	Intersport-TV	7
Stage-TV	3	Intersport-TV	5	Media Markt-TV	6
Cinema-TV	2	Highway	3	MEDIMAX-TV	6
Intersport-TV	2	Media Markt-TV	3	UNICUM TV	5
Kaufland TV	2	Stage-TV	3	Stage-TV	3
Cinemaxx	1	Cinema-TV	2	Kaufland TV	2
DCLP	1	Kaufland TV	2	Videoboards	2
Hochschulwerbung	1	MEDIMAX-TV	2	Cinema-TV	1
Media Markt-TV	1	Cinemaxx	1	Cinemaxx	1
MEDIMAX-TV	1	DCLP	1	Hochschulwerbung	1
		Hochschulwerbung	1	Shopping Center	1
		Videoboards	1		

Tabelle 7: Anzahl Standorte in digitalen-Werbeträger-Netzen der ermittelten DooH-Empfehlung 18:00 - 24:00 Uhr

18:00 - 21:00		21:00 - 24:00	
Netze	**Anzahl**	**Netze**	**Anzahl**
TV-Wartezimmer	87	TV-Wartezimmer	62
Gastro Channel	19	Campus-TV	21
adpack	17	Gastro Channel	16
Active-TV	16	UNICUM TV	13
Campus-TV	15	adpack	12
EDEKA TV	12	Active-TV	10
Kaisers TV	10	EDEKA TV	10
UNICUM TV	10	Hairstyling-TV	9
Hairstyling-TV	9	Apotheken-TV	7
McDonalds-TV	9	Cash-TV	7
Cash-TV	8	Kaisers TV	7
Apotheken-TV	7	McDonalds-TV	7
Cinema Channel	7	King Channel	5
King Channel	7	Cinema Channel	4
Media Markt-TV	6	Saturn-TV	4
Saturn-TV	6	Airport Berlin	3
Intersport-TV	5	Intersport-TV	3
Cinema-TV	3	Stage-TV	3
Stage-TV	3	Cinema-TV	2
MEDIMAX-TV	2	Kaufland TV	2
Cinemaxx	1	Media Markt-TV	2
DCLP	1	MEDIMAX-TV	2
Hochschulwerbung	1	Cinemaxx	1
Kaufland TV	1	Hochschulwerbung	1

Es fällt auf, dass über alle Zeitschienen hinweg, die digitalen Werbeträger des Netzes „*TV-Wartezimmer*“ den größten Anteil der DooH-Empfehlung aufweisen. Dies lässt sich dadurch begründen, dass das Netz mit einer Anzahl von 239 Standorten in Berlin die größte Abdeckung aufweist. Im Vergleich dazu gibt es in Berlin, über alle Anbieter hinweg, 866 Standorte von digitalen Werbeträgern.

7.4 Diskussion und Bewertung

Ziel dieser Arbeit war es zu überprüfen, ob mit Hilfe von Daten aus sozialen Netzwerken und maschineller Themenextraktion Zielgruppen-informationen gewonnen werden können, die dabei helfen digitale Außenwerbung zu planen. Es hat sich gezeigt, dass der hier gewählte Ansatz, die Textbeiträge mit Hilfe von LDA zu clustern, prinzipiell gut funktioniert. Leider sind die Ergebnisse, die hier gewonnen werden konnten, nicht so gut wie erhofft. Die Wort-Zusammensetzungen die mit LDA ermittelt worden sind, sind inhaltlich nicht in einem hohen Maße schlüssig, lassen jedoch eindeutige Oberthemen-Tendenzen erkennen. Dies lässt sich mit der geringen Anzahl an auswertbaren Textbeiträgen/Tweets erklären. Mit 3.591 Textbeiträgen, in der Zeitschiene mit der geringsten Anzahl an verwertbaren Textbeiträgen, ist die Datengrundlage zu dünn um inhaltlich sehr gute Ergebnisse erreichen zu können. Es hat sich gezeigt, dass Twitter in Deutschland nicht die ideale Datengrundlage für solche Analysen bietet. Um eine aussagekräftigere Mediaplanung erstellen zu können, wäre es zu empfehlen, die hier erarbeiteten Prozesse auf andere Datengrundlagen zu transferieren oder die hier gewonnen Ergebnisse mit weiteren Informationen anzureichern. Jede Mediaplanung ist individuell und bestimmt durch ihre konzeptionelle Rahmenbedingungen wie z.B. das Werbegebiet, das zu bewerbende Produkt, das Motiv, die erwünschte Werbewirkung usw. So ist z.B. „*Wartezimmer-TV*“ in allen Zeitschienen das Netz, mit der höchsten Anzahl an Standorten, jedoch beinhaltet das Netz nur digitale Werbeträger, die ausschließlich im Innenbereich von Arztpraxen aufgestellt sind. Hier können demnach nur wenig bis keine Blickkontakte durch Passanten erzielt werden. Die Entscheidung, ob sich die hier ermittelten Standorte für eine Werbekampagne eignen, muss immer individuell getroffen werden. Trotz der geringen Datengrundlage bieten die Ergebnisse jedoch eine gute Grundlage um neue Planungsansätze zu entwickeln. Dies bedarf natürlich immer der Bereitschaft des Kunden, neue und moderne Wege zu gehen. Die Netze, die hier für die Zielgruppe „Mode/Fashion-Interessierte“ identifiziert wurden, sind nicht die Netze, die ein Kunde intuitiv präferieren würde. So stehen z.B. „*Gastro Channel*“ oder „*Campus-TV*“ auf den ersten Blick nicht direkt mit Mode/Fashion in Verbindung. Jedoch lässt sich durch die hier durchgeführten Analysen eine räumliche und inhaltliche Verbindung herstellen welche neue Planungsansätze und neue Sichtweisen ermöglichen.

Quellenverzeichnis

Abney, S. (1997), "Part-of-Speech Tagging and Partial Parsing", in Young, S. and Bloothooft, G. (Eds.), *Corpus-Based Methods in Language and Speech Processing*, *Text, Speech and Language Technology*, Vol. 2, Springer, Dordrecht, pp. 118–136.

Anselin, L. (1993), *The Moran scatterplot as an ESDA tool to assess local instability in spatial association*, Regional Research Institute, West Virginia University Morgantown, WV.

Anselin, L. (1995), "Local Indicators of Spatial Association-LISA", *Geographical Analysis*, Vol. 27 No. 2, pp. 93–115.

"API Rate Limits — Twitter Developers", available at: https://dev.twitter.com/rest/public/rate-limiting (accessed 4 September 2017).

Blei, D.M. (2012), "Probabilistic topic models", *Communications of the ACM*, Vol. 55 No. 4, p. 77.

Blei, D.M., Ng, A.Y. and Jordan, M.I. (2003), "Latent Dirichlet Allocation", *J. Mach. Learn. Res.*, Vol. 3, pp. 993–1022.

Casella, G. and George, E.I. (1992), "Explaining the Gibbs Sampler", *The American Statistician*, Vol. 46 No. 3, p. 167.

Cheng, Z., Caverlee, J. and Lee, K. (2010), "You Are Where You Tweet: A Content-based Approach to Geo-locating Twitter Users", in *Proceedings of the 19th ACM International Conference on Information and Knowledge Management*, ACM, New York, NY, USA, pp. 759–768.

"CyHunspell 1.2.0 Python Package Index", available at: https://pypi.python.org/pypi/CyHunspell (accessed 4 September 2017).

DMI Digital Media Institute UG (2016), "Digital out of home", available at: http://www.dmi-org.com/.

EMC Corporation (2017), "EMC Privacy Index", available at: https://www.emc.com/campaign/privacy-index/index.htm.

Fachverband Außenwerbung e.V. (2017), "Plakatwerbung", available at: https://www.faw-ev.de/out-of-home-medien/plakatwerbung/.

"Flickr: Der Flickr Entwickler-Leitfaden - API", available at: https://www.flickr.com/services/developer/api/ (accessed 4 September 2017).

"gensim: topic modelling for humans" (2014), available at: https://radimrehurek.com/gensim/models/ldamallet.html (accessed 4 September 2017).

"gensim: topic modelling for humans" (2017a), available at: https://radimrehurek.com/gensim/tut1.html (accessed 4 September 2017).

"gensim: topic modelling for humans" (2017b), available at: https://radimrehurek.com/gensim/ (accessed 4 September 2017).

"Graph API /user/feed - Dokumentation - Facebook for Developers", available at: https://developers.facebook.com/docs/graph-api/reference/v2.6/user/feed (accessed 4 September 2017).

Grefenstette, G. and Tapanainen, P. (1997), "What is a word, What is a sentence? Problems of Tokenization".

Griffiths, T.L. and Steyvers, M. (2004), "Finding scientific topics", *Proceedings of the National Academy of Sciences*, Vol. 101 No. Supplement 1, pp. 5228–5235.

"Hunspell: About" (2017), available at: http://hunspell.github.io/ (accessed 4 September 2017).

Lewandowski, D. (2005), *Web Information Retrieval: Technologien zur Informationssuche im Internet*, Zugl.: Düsseldorf, Univ., Diss., 2005, *Reihe Informationswissenschaft der DGI*, Vol. 7, Dt. Ges. für Informationswiss. und Informationspraxis, Frankfurt am Main.

"LibreOffice/dictionaries", available at: https://github.com/LibreOffice/dictionaries (accessed 4 September 2017).

"MALLET homepage", available at: http://mallet.cs.umass.edu/ (accessed 4 September 2017).

Mimno, D., Wallach, H.M., Naradowsky, J., Smith, D.A. and McCallum, A. (2009), "Polylingual Topic Models", in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 2 - Volume 2*, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 880–889.

"Natural Language Toolkit — NLTK 3.2.4 documentation" (2017), available at: http://www.nltk.org/ (accessed 4 September 2017).

"Platform Policy - Instagram", available at: https://www.instagram.com/about/legal/terms/api/?hl=de (accessed 4 September 2017).

PostGIS, D. (2017), "PostGIS — Spatial and Geographic Objects for PostgreSQL", available at: http://postgis.net/ (accessed 4 September 2017).

"PostgreSQL + Python | Psycopg", available at: http://initd.org/psycopg/ (accessed 4 September 2017).

"PostgreSQL: About", available at: https://www.postgresql.org/about/ (accessed 4 September 2017).

Rajaraman, A., Leskovec, J. and Ullman, J.D. (2014), *Mining of Massive Datasets,* Version 2.0, Cambridge University Press, New York, N.Y., Cambridge.

"TextBlob: Simplified Text Processing — TextBlob 0.13.0 documentation" (2017), available at: https://textblob.readthedocs.io/en/dev/ (accessed 4 September 2017).

"textblob-de 0.4.2 Python Package Index", available at: https://pypi.python.org/pypi/textblob-de (accessed 4 September 2017).

"The Search API — Twitter Developers", available at: https://dev.twitter.com/rest/public/search (accessed 4 September 2017).

Tobler, W.R. (1970), "A Computer Movie Simulating Urban Growth in the Detroit Region", *Economic Geography*, Vol. 46 No. sup1, pp. 234–240.

"Über Flickr", available at: https://www.flickr.com/about (accessed 4 September 2017).

Wallach, H.M. (2006), "Topic modeling", in Cohen, W. (Ed.), *Proceedings of the 23rd international conference on Machine learning, Pittsburgh, Pennsylvania*, ACM, New York, NY, pp. 977–984.

Watson, D.F. and Philip, G.M. (1985), "A refinement of inverse distance weighted interpolation", *Geo-processing*, Vol. 2 No. 4, pp. 315–327.

"wooorm/gemoji", available at: https://github.com/wooorm/gemoji/blob/master/support.md (accessed 4 September 2017).

Zhao, W.X., Jiang, J., Weng, J., He, J., Lim, E.-P., Yan, H. and Li, X. (2011), "Comparing Twitter and Traditional Media Using Topic Models", in Clough, P., Foley, C., Gurrin, C., Jones, G.J.F., Kraaij, W., Lee, H. and Mudoch, V. (Eds.), *Advances in Information Retrieval*, *Lecture Notes in Computer Science*, Vol. 6611, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 338–349.

Zheng Yang, Jingfang Xu and Xing Li (2011), "Data Selection for User Topic Model in Twitter-Like Service", *2011 IEEE 17th International Conference on Parallel and Distributed Systems*, pp. 847–852.

8. Anhang

Anhang 1: Quellcode

main.py

```
# -*- coding: iso-8859-1 -*-

'''
Created on 18.03.2017
'''
from main_settings import *
from db_functions import *
from lda_functions import *
import time

def replaceUnicodeEmojiconsWithTextInTexts(workingTable):
    unicodeTexts = selectUnicodeEmjiconsText()

    for unicodeText in unicodeTexts:
        if unicodeText[1] == None:
            unicodeEmoji = ''
        else:
            unicodeEmoji = unicodeText[1]

        if unicodeText[0] == None:
            emojiText = ''
        else:
            emojiText = unicodeText[0]

        replaceUnicodeEmojiWithText(workingTable, unicodeEmoji,
        emojiText)


def normalizeTexts(workingTable):
    print ("   ...Replace Unicode Emojicons with Description from Texts")
    replaceUnicodeEmojiconsWithTextInTexts(workingTable)

    print ("   ...Delete HTTPS-Links form Texts")
    deleteHTTPSFromTexts(workingTable)

    print ("   ...Delete Hashtags #")
    deleteHastagsFromTexs(workingTable)

    print ("   ...Delete line breaks form Texts")
    replaceLineBreaksFromTexts(workingTable)

    addmittedChars = "\sa-zäöüßA-ZÄÖÜ"
    print ("   ...Delete all characters except from: '" +
    addmittedChars + "'")

    replacePermittedCharactersFromTexts(addmittedChars, workingTable)
```

```
    if ignoreBlacklist == False:
        print ("   ...Delete words from blacklist")
        removeWordsFromList(workingTable, blacklist)

    print ("   ...Delete Text from user-blacklist")
    deleteTextsFromBlacklistUser(workingTable, blacklist_user)

    print ("   ...Delete Texts wich are equal more then 3 times")
    deleteTripleTexts(workingTable)

    wordLenght = 2
    print ("   ...Delete words with a lenght less or equal than " +
    str(wordLenght))

    removeWordsLessThanLenghtOf(workingTable, wordLenght)

    print ("   ...Delete unessesary spaces")
    removeUnessesarySpacesFromTexts(workingTable)

    print ("   ...Delete lines which are now empty")
    deleteEmptyTexts(workingTable)

def removeBotTexts(workingTable, identiferBB):
    botTable = workingTable + "_bots"

    print ("   ...Save Bot users")
    identifyBotUser(workingTable, botTable, identiferBB)

    print ("   ...Delete all Texts from saved Bot users")
    deleteBotTexts(workingTable, botTable)

def prepareTextsForAnalyzation(workingTable, identiferBB, deleteBots,
    time_greater, time_lower):

    createTableForBBoxDocs(workingTable, True)
    print("...Table '" + workingSchema + "." + workingTable +
    "' created" )

    print ("...Copy data into table")
    copyDataIntoWorkingTable(bboxID, workingTable, time_greater,
    time_lower)

    if deleteBots == True:
        print ("...Begin deleting Bots Texts...")
        removeBotTexts(workingTable, identiferBB)

    print ("...Begin normalizing Texts...")
    normalizeTexts(workingTable)

    print ("...Add new unique column for identification")
    addAnalysationIDColunmn(workingTable)

def converteTupelListToStringList(tupelList):
    string_list = [doc[0] for doc in tupelList]
    return string_list

def convertDocTopicsToCSV(ldaModel, csvFileName, topicNum):
    docTopics = ldaModel.load_document_topics()
    file = open(workingPath + csvFileName, "w"
```

```
    line = "analyzation_id"


    # First line - Column Header
    for i in range(topicNum):
        line += ";{}".format("topic_" + str(i))
    file.write(line + "\n")

    i = 1
    for doc in docTopics:
        line = str(i)
        for item in doc:
            # 2. Value = Probability of document to topic
            _, topic_pb = item
            line += ";{}".format(topic_pb)
        file.write(line + "\n")
        i = i +1
    file.close()

def convertTopicsWordsToCSV(ldaModel, csvFileName, numOfWords):
    topics = ldaModel.print_topics(num_topics=-1, num_words=numOfWords)
    file = open(workingPath + csvFileName, "w")

    line = "topic_id;probability;word"
    file.write(line + "\n")

    for topic in topics:
        topic_id, topic_string = topic

        for wordProb in topic_string.split("+"):
            prob = wordProb.split("*")[0].replace(' ', '')
            word = wordProb.split("*")[1].replace('"', '').
            replace(' ', '')

            line = "{};{};{}".format(topic_id, prob, word)
            file.write(line + "\n")

def loadLDAResultsIntoDB(ldaModel,workingTable, topicNum):
    workingTable =  "tn_" + str(topicNum) + "_" + workingTable

    print("...Save Topics of Texts to CSV-File")
    csvMessFileName = workingTable + "_messages.csv"
    tableTopicsWords = workingTable + "_messages"
    convertDocTopicsToCSV(ldaModel, csvMessFileName, topicNum)

    print("...Create table:'"+ tableTopicsWords +"' for Topics of Texts")
    createTableForMessageTopics(tableTopicsWords, True, topicNum)

    print("...Copy data into table: '"+ tableTopicsWords)
    copyFromTXTFileIntoTable(tableTopicsWords, workingPath +
    csvMessFileName, ';')


    print("...Save Words/Probability of Topics to CSV-File")
    csvTopicsFileName = workingTable + "_topics.csv"
    tableTopics = workingTable + "_topics"
    convertTopicsWordsToCSV(ldaModel, csvTopicsFileName,
    numberWordsOfTopic)
```

```
    print("...Create Table:'"+ tableTopics +
    "' for Words/Probability of Topics")

    createTableForTopicResults(tableTopics, True)

    print("...Copy data into table: '"+ tableTopics)
    copyFromTXTFileIntoTable(tableTopics, workingPath +
    csvTopicsFileName, ';')

def main():
    print ("Topic Analyzer is started")

    identiferBB = getWorkingTableIdentifier(bboxID)
    tableIdentifier = workingTableIdentifier
    workingTable = tableIdentifier + identiferBB

    iTime_greater = 0
    iTime_lower = split_hour

    while iTime_lower <= 24:

        if split_hour != -1:
            time_greater = str(iTime_greater) + ":00"
            workingTable = tableIdentifier + identiferBB + '_' +
            str(iTime_greater)

            time_lower = str(iTime_lower) + ":00"
            workingTable = workingTable + '_' + str(iTime_lower)
        else:
            time_greater = ""
            time_lower = ""

        print ("You are going to analyze social media posts of the city:
        '" + identiferBB + "'")
        print ("Only Text which are posted between " + time_greater + "
        and " + time_lower)

        if prepare_data == True:
            print ("The texts are going to re-prepare for the
            analyzation")

            print ("Begin Text preparation...")
            prepareTextsForAnalyzation(workingTable, identiferBB,
            delete_bots, time_greater, time_lower)

        if loadFile == '':
            print ("Select Texts for Analysation")
            docs_tupel = selectTextsForTopicAnalysation(workingTable)

            docs_strings = converteTupelListToStringList(docs_tupel)

            start = time.time()

            print ("Clean Texts...")
            clean_docs = docToCleanDoc(docs_strings)

            print ("Calculate LDA-Model...")
            ldaModel_50 = determineTopics(clean_docs, 50)
```

```
            end = time.time()
            print("Time used for calculating LDA-Model: {}
            sek.".format(end - start))
        else:
            print ("Load LDA-Model from file '" + loadFile + "'")
            ldaModel = loadLDAModelFile(workingPath + loadFile)

        loadLDAResultsIntoDB(ldaModel_50,workingTable, 50)

        if loadFile == '':
            saveFile = "model_" + "tn_" + str(50) + "_" + workingTable +
            ".model"

            print ("LDA-Model was saved")

        if split_hour == -1:
            break
        else:
            iTime_greater = iTime_greater + split_hour
            iTime_lower = iTime_lower + split_hour

    print("Done!")

if __name__ == '__main__':
    main()
```

lda_functions.py

```
# -*- coding: iso-8859-1 -*-

'''
Created on 19.03.2017
'''

from nltk.corpus import stopwords
import string
from gensim.models.wrappers import LdaMallet
from gensim import corpora
from textblob import blob
from textblob_de.lemmatizers import PatternParserLemmatizer
from textblob import Word
from hunspell import Hunspell

def removeMutatedVowel(word):
    word = word.replace('ä', 'ae')
    word = word.replace('ö', 'oe')
    word = word.replace('ü', 'ue')
    word = word.replace('ß', 'ss')
    return word

def addMutatedVowel(word):
    word = word.replace('ae', 'ä')
    word = word.replace('oe', 'ö')
    word = word.replace('ue', 'ü')
    word = word.replace('ss', 'ß')
    return word

def spellCorrection(corr_language, correction_string):
    if corr_language == 'en':
        h = Hunspell('en_GB',
        "C:/Python353/Lib/site-packages/dictionaries")
    elif corr_language == 'de':
        h = Hunspell('de_DE_umlaut',
        "C:/Python353/Lib/site-packages/dictionaries")
    else:
        return correction_string

    spell_correct = ''
    for word in correction_string.split():
        if corr_language == 'de':
            word = removeMutatedVowel(word)
        if h.spell(word) == False:
            list_suggest = h.suggest(word)
            if len(list_suggest) > 0:
                if list_suggest[0] != '':
                    word = list_suggest[0].lower()

        if corr_language == 'de':
            word = addMutatedVowel(word)
```

```
    if spell_correct != '':
            spell_correct = spell_correct + ' '
        spell_correct = spell_correct + word

    return spell_correct

def LemmaCorrection(corr_language, correction_string):
    if corr_language == 'en':
        lemmatized = " ".join(Word(word).lemmatize() for word in
        correction_string.split())
    elif corr_language == 'de':
        lemma = PatternParserLemmatizer()
        lemma_list = " ".join(lemma.lemmatize(word)[0][0] for word in
        correction_string.split())
        lemmatized = " ".join(word.lower() for word in
        lemma_list.split())
    else:
        lemmatized = correction_string

    return lemmatized

def clean(doc, i):
    try:
        language = blob.TextBlob(doc).detect_language()
    except:
        language = 'errorDetect'

    if language == 'de':
        stop = set(stopwords.words('german'))
    elif language == 'en':
        stop = set(stopwords.words('english'))
    elif language == 'es':
        stop = set(stopwords.words('spanish'))
    elif language == 'tr':
        stop = set(stopwords.words('turkish'))
    elif language == 'fr':
        stop = set(stopwords.words('french')) )
    else:
        language = 'notFound'

    if (language != 'errorDetect') and (language != 'notFound'):
        exclude = set(string.punctuation)

        if (language == 'de') or (language == 'en'):
            spell_correct = spellCorrection(language, doc)
            lemma_correct = LemmaCorrection(language, spell_correct)
        else:
            lemma_correct = stem_free
    else:
        lemma_correct = doc

    stop = set(stopwords.words('german'))
    stop_de = " ".join([i for i in lemma_correct.lower().split()
    if i not in stop])

    stop = set(stopwords.words('spanish'))
    stop_es = " ".join([i for i in stop_de.lower().split()
    if i not in stop])
```

```
    stop = set(stopwords.words('french'))
    stop_fr = " ".join([i for i in stop_es.lower().split()
    if i not in stop])

    stop = set(stopwords.words('english'))
    stop_en = " ".join([i for i in stop_fr.lower().split()
    if i not in stop])

    # remove too short words again, because of corrections
    no_short = " ".join([word for word in stop_en.lower().split()
    if len(word) > 2])

    print ('...Doc: [' + str(i) + ']')

    return no_short

def determineTopics(docs_clean, topic_num):
    # Creating the term dictionary of our courpus, where every unique
    term is assigned an index.
    dictionary = corpora.Dictionary(docs_clean)

    # Converting list of documents (corpus) into Document Term Matrix
    using dictionary prepared above.
    doc_term_matrix = [dictionary.doc2bow(doc) for doc in docs_clean]

    ldaModel = malletLDA_Model(doc_term_matrix, dictionary, topic_num)

    return ldaModel

def docToCleanDoc(docs):
    docs_clean = [clean(doc, i).split() for i, doc in enumerate(docs)]

    return docs_clean

def malletLDA_Model(doc_term_matrix, dictionary, topic_num):
    ldaModel = LdaMallet("C:/mallet208/bin/mallet",
    corpus=doc_term_matrix, num_topics=topic_num, id2word=dictionary)

    return ldaModel

def loadLDAModelFile(FileLocation):
    return gensim.models.wrappers.LdaMallet.load(FileLocation)

def printDocumentTopicsMallet(ldaModel):
    docTopics = ldaModel.load_document_topics()
    for doc in docTopics:
        print(doc)

def printWordsOfTopicsMallet(ldaModel, numWordsTopic):
    formatted_topics = ldaModel.print_topics(num_topics=-1,
    num_words=numWordsTopic)

    for formatted_topic in formatted_topics:
        print(formatted_topic)
```

db_functions.py

```
# -*- coding: iso-8859-1 -*-

'''
Created on 18.03.2017
'''

import psycopg2 as dbapi
from main_settings import *
from numpy.core.defchararray import lower


def initDBConnection(host, dbName, user, password, port):
    db = dbapi.connect(host=host,database=dbName, user=user,
    password=password, port=port)

    cur = db.cursor()
    return cur, db

def openSQLQuery(sSQL):
    cur, db = initDBConnection(host, dbName, user, password, port)
    cur.execute(sSQL)
    return cur.fetchall()

def execSQLQuery(sSQL):
    cur, db = initDBConnection(host, dbName, user, password, port)
    cur.execute(sSQL)
    db.commit()

def tabelExists(schema, table):
    sSQL = """ Select table_name
            from information_schema.tables
            where table_schema = '"""+ schema +"""' and
            table_name = '""" + table + """' """

    queryResult = openSQLQuery(sSQL)

    if len(queryResult) == 0:
        return False
    else:
        return True

def getBBoxParams(bboxID):
    sSQL = """ Select id, name
            from """ + sourceSchema + """.boundbox
            where id = """ + str(bboxID)

    return openSQLQuery(sSQL)

def getWorkingTableIdentifier(bboxID):
    bboxList = getBBoxParams(bboxID)

    # First line, because result only one BoundingBox
    bbox = bboxList[0]
```

```
    # column: name
    identiferBB = bbox[1].lower()

    return identiferBB

def createTableForBotUser(botTable):
    sSQLCreate = """CREATE TABLE """ + workingSchema + "." +
    botTable + """
            (
              user_id character varying NOT NULL,
              CONSTRAINT """ + botTable + """_pkey PRIMARY KEY (user_id)
            )
            WITH (
              OIDS=FALSE
            );
            ALTER TABLE """ + workingSchema + "." + botTable + """
              OWNER TO postgres;"""

    execSQLQuery(sSQLCreate)


def createTableForBBoxDocs(workingTable, deleteExistingTable):
    # if table already exits -> drop it
    if tabelExists(workingSchema, workingTable) == True:
        if deleteExistingTable == True:
            execSQLQuery("Drop Table " + workingSchema + "." +
            workingTable)
            print ("...Table for analyzing texts deleted")
        else:
            print ("Process is stoped")
            exit(0)

    sSQLCreate = """CREATE TABLE """ + workingSchema + "." +
    workingTable + """
                (
                  message_id bigint NOT NULL,
                  repost character varying(5),
                  date character varying(60),
                  user_id character varying(50),
                  username character varying(50),
                  text character varying,
                  tags character varying(300),
                  photo_url character varying(200),
                  geom_org geometry(Point,4326),
                  geom_utm geometry(Point,32618),
                  bbox_id integer,
                  text_clean character varying,
                  CONSTRAINT """ + workingTable +"""_pkey
                  PRIMARY KEY (message_id)
                )
                WITH (
                  OIDS=FALSE
                );
                ALTER TABLE """ + workingSchema + "." +
                workingTable + """
                  OWNER TO postgres;

                CREATE INDEX """ + workingTable + """_user_id_idx
                  ON """ + workingSchema + "." + workingTable + """
```

```
                    USING btree
                    (user_id COLLATE pg_catalog."default");"""

    execSQLQuery(sSQLCreate)

def createTableForTopicResults(tableName, deleteExistingTable):
    # if table already exits -> drop it
    if tabelExists(workingSchema, tableName) == True:
        if deleteExistingTable == True:
            execSQLQuery("Drop Table " + workingSchema + "." + tableName)
            print ("...Table for analyzing texts deleted")
        else:
            print ("Process is stoped")
            exit(0)

    sSQLCreate = """CREATE TABLE """ +
    workingSchema + "." + tableName + """
                (
                  topic_id INTEGER,
                  probability numeric,
                  word character varying(100)
                )
                WITH (
                  OIDS=FALSE
                );
                ALTER TABLE """ + workingSchema + "." + tableName + """
                  OWNER TO postgres;"""

    execSQLQuery(sSQLCreate)

def createTableForMessageTopics(tableName, deleteExistingTable,
topicNum):
    # if table already exits -> drop it
    if tabelExists(workingSchema, tableName) == True:
        if deleteExistingTable == True:
            execSQLQuery("Drop Table " + workingSchema + "." + tableName)
            print ("...Table for analyzing texts deleted")
        else:
            print ("Process is stopped")
            exit(0)

    sSQLCreate = """CREATE TABLE """ +
    workingSchema + "." + tableName + """(
                    analyzation_id integer """

    for i in range(topicNum):
        sSQLCreate = sSQLCreate + ", topic_" + str(i) + " numeric "

    sSQLCreate = sSQLCreate + """ )
                WITH (
                  OIDS=FALSE
                );
                ALTER TABLE """ + workingSchema + "." + tableName + """
                  OWNER TO postgres;"""

    execSQLQuery(sSQLCreate)
```

```
def copyDataIntoWorkingTable(bboxID, workingTable, time_greater,
time_lower):
    sSQL = """ Insert into """ + workingSchema + "." + workingTable + """
            Select min(message_id) as message_id, repost, date, user_id,
            username, text, tags, photo url,
            t.geom_org, t.geom_utm, bb.id as bbox_id, text as text_clean
            From """ + sourceSchema + """.boundbox bb,
            """ + sourceSchema + """."""" + sourceTable + """ t
            where bb.id = """ + str(bboxID) + """ and
            st_contains(bb.geom_org, t.geom_org) = True """

    if (time_greater != ''):
        sSQL = sSQL + " and cast(date as time) >= cast('"+
        time_greater +"' as time) "

    if (time_lower != ''):
        sSQL = sSQL + " and cast(date as time) <= cast('"+
        time_lower +"' as time) "


    sSQL = sSQL + " Group by repost, date, user_id, username, text, tags,
    photo_url, t.geom_org, t.geom_utm, bb.id "

    execSQLQuery(sSQL)

def foundHTTPSInTexts(workingTable):
    sSQL = """ Select message_id from """ +
    workingSchema + "." + workingTable + """
            where text_clean like '%https:%'
            limit 1 """

    queryResult = openSQLQuery(sSQL)

    if len(queryResult) == 0:
        return False
    else:
        return True

def copyFromTXTFileIntoTable(targetTable, filePathName, delimiter):
    sSQL = """ Copy """ + workingSchema + "." + targetTable + """
            From '""" + filePathName + """'
            delimiter '""" + delimiter + """'
            csv header
            encoding 'LATIN9';"""

    execSQLQuery(sSQL)

def deleteHTTPSFromTexts(workingTable):
    # As long a https-link is found, delete them
    while foundHTTPSInTexts(workingTable) == True:
        sSQL = """Update """ + workingSchema + "." + workingTable + """
                set text_clean = case when position(' ' in
                substring(text_clean, position('https:' in text_clean),
                char_length(text_clean))) = 0
                then replace(text_clean, substring(text_clean,
                position('https:' in text_clean),
                char_length(text_clean)), '')
                else replace(text_clean,substring(substring(text_clean,
                position('https:' in text_clean),
```

```
                char_length(text_clean)), 1, position(' ' in
                substring(text_clean, position('https:' in text_clean),
                char_length(text_clean)))), '') end
                where text_clean like '%https:%';"""

        execSQLQuery(sSQL)

def deleteHastagsFromTexs(workingTable):
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = replace(text_clean, '#', ' ')
            where text_clean like '%#%';"""

    execSQLQuery(sSQL)

def replaceLineBreaksFromTexts(workingTable):
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = regexp_replace(text_clean,
            E'[\\n\\r]+', ' ', 'g' )
            where (text_clean like '%'||chr(10)||'%') or
            (text_clean like '%'||chr(13)||'%');"""

    execSQLQuery(sSQL)

def replacePermittedCharactersFromTexts(addmittedChars, workingTable):
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = regexp_replace(text_clean,
            '[^"""+ addmittedChars +"""]+', ' ', 'g')"""

    execSQLQuery(sSQL)

def removeUnessesarySpacesFromTexts(workingTable):
    # Remove spaces which are caused by the previous functions
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = trim(regexp_replace(text_clean,
            '\s+', ' ', 'g'))
            where text_clean like '%  %'"""

    execSQLQuery(sSQL)

    # Remove unessesary spaces from beginning an end
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = trim(text_clean)
            where text_clean like '% ' or text_clean like ' %'"""

    execSQLQuery(sSQL)

def removeWordsLessThanLenghtOf(workingTable, wordLenght):
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = regexp_replace(text_clean, $$\y\w{1,"""+
            str(wordLenght) +"""}\y$$, '', 'g' )"""

    execSQLQuery(sSQL)

def removeWordsFromList(workingTable, wordList):
    fromWord = "text_clean"
    for word in wordList:
        string_remove = "replace(lower("+ fromWord +"), '"+
        word.lower() +"', '')"
        fromWord = string_remove
```

```
    sSQL = """Update """ + workingSchema + "." + workingTable + """
            set text_clean = """ + string_remove + """ where """

    for i, word in enumerate(wordList):
        if i != 0:
            sSQL = sSQL + " or "

        sSQL = sSQL + " lower(text_clean) like '%" + word.lower() + "%' "

    execSQLQuery(sSQL)

def replaceUnicodeEmojiWithText(workingTable, unicode, emojiText):
    sSQL = """Update """ + workingSchema + "." + workingTable + """
               set text_clean = replace(text_clean, '""" + unicode +
               """', ' """ + emojiText + """ ')
               Where text_clean like '%""" + unicode + """%' """

    execSQLQuery(sSQL)

def deleteTextsFromBlacklistUser(workingTable, userList):
    sSQL = """Delete from """ + workingSchema + "." + workingTable + """
            where """

    for i, user in enumerate(userList):
        if i != 0:
            sSQL = sSQL + " or "

        sSQL = sSQL + " user_id = '" + user + "'"

    execSQLQuery(sSQL)

def deleteTripleTexts(workingTable):
    sSQL = """Delete from """ + workingSchema + "." + workingTable + """
            where (text_clean, user_id, username) in (
                Select text_clean, user_id, username
                from """ + workingSchema + "." + workingTable + """
                group by text_clean, user_id, username
                having count(*) > 2
                order by username, count(*))"""

    execSQLQuery(sSQL)

def deleteEmptyTexts(workingTable):
    # Delete lines, which are empty because of not interpretable
    characters
    sSQL = """Delete from """ + workingSchema + "." + workingTable + """
            where text_clean = '' or text_clean = ' ';"""

    execSQLQuery(sSQL)

def identifyBotUser(workingTable, botTable, identiferBB):
    # if table already exits -> drop it
    if tabelExists(workingSchema, botTable) == True:
        execSQLQuery("Drop Table " + workingSchema + "." + botTable)
        print ("    ...Table for Bot users deleted")

    createTableForBotUser(botTable)
```

```
    sSQL = """ Insert into """ + workingSchema + "." + botTable + """
            Select distinct user_id
            from(
                Select count_word_in_massegaes/messages as mw,
                k.user id, messages
                from(
                    Select user_id, word, sum(words_per_message) as
                    words_abs, cast(count(*) as real) as
                    count_word_in_massegaes,
                    string_agg(cast(message_id as varchar),
                    ';' Order by message_id) as message_ids
                    from(
                        Select user_id, message_id, word, count(*) as
                        words_per_message
                        from(
                            Select user_id, message_id,
                            UPPER(regexp_split_to_table
                            (regexp_replace(text_clean,
                            '[^\s''a-zäöüßA-ZÄÖÜ0-9]+', '', 'g'),
                            ' ')) as word, text_clean
                            from """ + workingSchema + "." +
                            workingTable + """
                            where text_clean not like '%I''m at%')i
                        where trim(word) <> ''
                        Group by user_id, message_id, word)ii
                    Group by user_id, word
                    order by count_word_in_massegaes)k,
                    (Select user_id, cast(count(message_id) as real) as
                     messages
                    from """ + workingSchema + "." + workingTable + """
                    Group by user_id
                    Order by cast(user_id as numeric))m
                where k.user_id = m.user_id and
                count_word_in_massegaes/messages >= """ +
                str(ratioOfRedundancy) + """ and
                messages >= """ + str(minBotPosts) + """
                Order by messages desc, mw desc)iii;"""

    execSQLQuery(sSQL)

def deleteBotTexts(workingTable, botTable):
    sSQL = """ Delete from """ + workingSchema + "." + workingTable + """
            where message_id in (
                Select message_id
                from """ + workingSchema + "." + botTable + """ b,
                """ + workingSchema + "." + workingTable + """ d
                where d.user_id = b.user_id
                order by b.user_id, text_clean);"""

    execSQLQuery(sSQL)

def addAnalysationIDColunmn(workingTable):
    sSQL = "ALTER TABLE " + workingSchema + "." + workingTable + " ADD
           COLUMN "+ analyzeID +" serial;"
    execSQLQuery(sSQL)
```

```
def selectTextsForTopicAnalysation(workingTable):
    sSQL = """Select text_clean from """ +
    workingSchema + "." + workingTable + """
            Order by """ + analyzeID

    return openSQLQuery(sSQL)

def selectUnicodeEmjiconsText():
    sSQL = "Select namen || ' ' || tags as namen, unicode from " +
    sourceSchema + ".unicode_emojicons"

    return openSQLQuery(sSQL)
```

Anhang 2: Extrahierte Themencluster

Zeitschiene: 00:00 – 03:00 Uhr

0	0.041	berl	1	0.037	zoo	2	0.061	mal	3	0.040	favourite	4	0.204	night
0	0.029	alemania	1	0.036	tit	2	0.035	gehen	3	0.026	fire	4	0.102	good
0	0.022	hoy	1	0.034	girl	2	0.029	schon	3	0.025	case	4	0.028	late
0	0.018	solo	1	0.027	woman	2	0.028	sehen	3	0.024	part	4	0.019	soda
0	0.015	victory	1	0.020	winter	2	0.027	kommen	3	0.019	play	4	0.015	monday
5	0.277	photo	6	0.054	face	7	0.040	checkpoint	8	0.053	christmas	9	0.052	people
5	0.197	posted	6	0.050	rock	7	0.033	charlie	8	0.041	charlottetown	9	0.043	holiday
5	0.058	drinking	6	0.037	fashion	7	0.020	cold	8	0.029	christmastime	9	0.024	weekend
5	0.037	video	6	0.037	hard	7	0.017	check	8	0.028	friday	9	0.018	forget
5	0.019	poster	6	0.032	style	7	0.014	sleep	8	0.023	sunset	9	0.018	picture
10	0.091	food	11	0.058	amazing	12	0.152	club	13	0.142	time	14	0.063	show
10	0.038	dinner	11	0.047	plat	12	0.037	party	13	0.075	live	14	0.042	thing
10	0.033	burger	11	0.034	cool	12	0.019	matrix	13	0.030	joy	14	0.037	bahnhof
10	0.023	white	11	0.030	centre	12	0.018	gretchen	13	0.022	feel	14	0.033	make
10	0.023	porn	11	0.025	nosy	12	0.018	watergate	13	0.020	tear	14	0.027	sound
15	0.129	hotel	16	0.114	travel	17	0.038	arena	18	0.108	happy	19	0.133	amp
15	0.034	home	16	0.036	europa	17	0.030	marathon	18	0.071	friend	19	0.049	beer
15	0.030	hour	16	0.029	trip	17	0.021	rot	18	0.052	birthday	19	0.017	vegan
15	0.028	hostel	16	0.023	view	17	0.018	mercedes	18	0.042	house	19	0.016	dog
15	0.022	cat	16	0.021	tower	17	0.016	running	18	0.028	boy	19	0.016	restaurant
20	0.041	station	21	0.050	place	22	0.107	city	23	0.208	bar	24	0.082	great
20	0.040	star	21	0.041	home	22	0.028	church	23	0.038	lindbergh	24	0.033	guy
20	0.040	hauptmann	21	0.033	kind	22	0.019	tonight	23	0.037	nice	24	0.031	summer
20	0.029	autobahn	21	0.025	tonight	22	0.017	bit	23	0.033	cocktail	24	0.023	lovely
20	0.022	awesome	21	0.022	set	22	0.017	future	23	0.027	panorama	24	0.020	job
25	0.042	mehr	26	0.042	spree	27	0.035	stag	28	0.034	berlim	29	0.054	mitre
25	0.039	geben	26	0.038	friedrich	27	0.035	neural	28	0.021	foto	29	0.038	berg
25	0.038	immer	26	0.030	team	27	0.034	stagecoach	28	0.020	mitte	29	0.033	garden
25	0.029	sagen	26	0.022	repost	27	0.034	building	28	0.016	mas	29	0.025	prenuptial
25	0.020	leben	26	0.019	tuesday	27	0.031	architecture	28	0.015	ser	29	0.022	botanical
30	0.059	back	31	0.064	light	32	0.044	super	33	0.166	art	34	0.035	red
30	0.032	tag	31	0.062	ale	32	0.028	head	33	0.102	street	34	0.029	made
30	0.030	welt	31	0.048	festival	32	0.024	night-life	33	0.053	photography	34	0.022	saturday
30	0.026	top	31	0.026	film	32	0.020	sunday	33	0.041	graffiti	34	0.021	trump
30	0.023	neü	31	0.023	restaurant	32	0.020	wedding	33	0.036	urban	34	0.020	elmer
35	0.169	wall	36	0.116	nuremberg	37	0.091	gut	38	0.142	day	39	0.187	love
35	0.109	east	36	0.035	armut	37	0.056	nacht	38	0.097	year	39	0.058	airport
35	0.096	gallery	36	0.020	repeat	37	0.027	kurfürstendamm	38	0.055	today	39	0.044	heart
35	0.088	side	36	0.018	making	37	0.026	heuen	38	0.030	week	39	0.022	tegel
35	0.016	world	36	0.013	nehmen	37	0.021	abend	38	0.027	tomorrow	39	0.018	eye
40	0.034	neukölln	41	0.068	park	42	0.077	alexander	43	0.094	memorial	44	0.074	party
40	0.030	coffee	41	0.056	palaeozaen	42	0.061	tor	43	0.029	europe	44	0.040	deutsch
40	0.029	perfect	41	0.035	kindergarten	42	0.049	flag	43	0.027	holocaust	44	0.039	sky
40	0.025	history	41	0.022	repertoire	42	0.044	gate	43	0.022	murdered	44	0.029	ehe
40	0.025	phone	41	0.021	kreuzberg	42	0.019	alex	43	0.020	europa	44	0.025	technocrat
45	0.077	music	46	0.091	museum	47	0.046	platz	48	0.046	danke	49	0.026	wild
45	0.055	life	46	0.038	world	47	0.040	alt	48	0.039	drink	49	0.022	igers
45	0.038	studio	46	0.037	long	47	0.024	boom	48	0.037	marketeer	49	0.020	beautiful
45	0.033	concert	46	0.035	dom	47	0.023	jahr	48	0.018	kraftwerk	49	0.020	wine
45	0.023	columbia	46	0.030	beautiful	47	0.021	neüs	48	0.016	paul	49	0.018	wuhlheide

Zeitschiene: 03:00 – 06:00 Uhr

0	0.050	week	1	0.100	home	2	0.157	day	3	0.143	art	4	0.103	schönefeld
0	0.036	show	1	0.058	life	2	0.062	happy	3	0.077	street	4	0.089	flughafen
0	0.036	fashion	1	0.049	live	2	0.059	people	3	0.059	house	4	0.048	rapport
0	0.034	work	1	0.037	christmas	2	0.038	air	3	0.038	urban	4	0.038	scheu
0	0.022	lab	1	0.023	market	2	0.026	heirate	3	0.028	spree	4	0.031	sex
5	0.032	audio	6	0.166	wall	7	0.051	life	8	0.043	checkpoint	9	0.113	bar
5	0.025	boom	6	0.067	gallery	7	0.037	mitre	8	0.033	charlie	9	0.064	lindbergh
5	0.020	polizei	6	0.058	side	7	0.037	make	8	0.031	set	9	0.047	panorama
5	0.020	alee	6	0.053	east	7	0.032	flu	8	0.031	experience	9	0.022	trip
5	0.017	visit	6	0.039	throwback	7	0.025	studio	8	0.024	friday	9	0.022	super
10	0.066	series	11	0.060	hotel	12	0.162	travel	13	0.071	lounge	14	0.094	memorial
10	0.061	photography	11	0.043	tonight	12	0.038	gram	13	0.028	alemania	14	0.031	europe
10	0.037	woman	11	0.038	technocrat	12	0.038	travelogue	13	0.028	struma	14	0.031	today
10	0.029	film	11	0.026	tresor	12	0.027	photographed	13	0.028	mussen	14	0.021	soviet
10	0.020	job	11	0.024	beryl	12	0.025	tit	13	0.025	dream	14	0.018	church
15	0.034	hauptmann	16	0.118	good	17	0.069	world	18	0.107	amp	19	0.045	station
15	0.029	nett	16	0.086	morning	17	0.050	flag	18	0.062	boy	19	0.040	view
15	0.029	rein	16	0.029	fun	17	0.047	international	18	0.042	berl	19	0.038	drink
15	0.021	green	16	0.029	stage	17	0.030	book	18	0.039	selfish	19	0.033	sound
15	0.018	story	16	0.019	crew	17	0.020	follow	18	0.026	ostbahnhof	19	0.033	charlottetown
20	0.056	buzz	21	0.226	love	22	0.120	gut	23	0.104	city	24	0.055	light
20	0.040	eppingen	21	0.035	rock	22	0.096	morgen	23	0.083	europa	24	0.044	cool
20	0.031	uhr	21	0.028	god	22	0.079	mal	23	0.075	amazing	24	0.026	joy
20	0.025	letzt	21	0.025	picture	22	0.051	heuen	23	0.026	hard	24	0.016	tit
20	0.025	wer	21	0.020	list	22	0.033	gehen	23	0.021	seit	24	0.016	sunglass
25	0.205	photo	26	0.033	state	27	0.039	video	28	0.114	year	29	0.105	friend
25	0.151	posted	26	0.031	police	27	0.023	humidity	28	0.016	company	29	0.041	atonal
25	0.016	bitch	26	0.021	king	27	0.020	alex	28	0.013	collection	29	0.036	kraftwerk
25	0.014	berlim	26	0.021	dead	27	0.017	performance	28	0.013	kurfürstendamm	29	0.031	awesome
25	0.011	karaoke	26	0.021	attack	27	0.014	andy	28	0.013	loved	29	0.031	face
30	0.061	bahnhof	31	0.042	early	32	0.153	party	33	0.161	time	34	0.034	weekend
30	0.027	main	31	0.025	bird	32	0.033	star	33	0.087	back	34	0.034	tor
30	0.027	winter	31	0.022	eye	32	0.033	dance	33	0.036	hour	34	0.029	wait
30	0.027	tour	31	0.019	day	32	0.025	halloween	33	0.034	long	34	0.026	fire
30	0.024	smile	31	0.019	miss	32	0.020	nosy	33	0.019	london	34	0.021	coming
35	0.027	jed	36	0.037	friedrich	37	0.289	airport	38	0.051	zoo	39	0.086	museum
35	0.027	amp	36	0.037	deutsch	37	0.119	tegel	38	0.051	bahn	39	0.036	dom
35	0.025	beim	36	0.029	centre	37	0.096	otto	38	0.033	olympiad	39	0.036	style
35	0.022	sahn	36	0.029	stagecoach	37	0.095	lilienthal	38	0.030	bvg	39	0.018	thing
35	0.015	part	36	0.024	building	37	0.051	schoolfellow	38	0.028	frühschicht	39	0.016	island
40	0.111	alexander	41	0.087	park	42	0.074	great	43	0.084	gate	44	0.177	night
40	0.037	tower	41	0.045	heart	42	0.042	tag	43	0.071	holiday	44	0.081	music
40	0.032	beer	41	0.040	click	42	0.032	girl	43	0.045	architecture	44	0.033	late
40	0.029	garden	41	0.020	flirt	42	0.027	mood	43	0.021	goodbye	44	0.033	hauptmann
40	0.021	step	41	0.018	kommen	42	0.025	top	43	0.016	brandenburg	44	0.028	making
45	0.043	summer	46	0.048	food	47	0.037	elmer	48	0.123	nuremberg	49	0.169	club
45	0.027	place	46	0.029	black	47	0.034	plat	48	0.025	memory	49	0.037	official
45	0.022	birthday	46	0.021	stresses	47	0.029	angela	48	0.020	bundestag	49	0.035	watergate
45	0.016	repeat	46	0.021	white	47	0.029	chancellor	48	0.020	town	49	0.028	sky
45	0.014	fun	46	0.021	porn	47	0.024	lieb	48	0.018	artist	49	0.028	arena

Zeitschiene: 06:00 – 09:00 Uhr

0	0.183	photo	1	0.052	ice	2	0.056	time	3	0.033	lounge	4	0.082	early
0	0.114	posted	1	0.044	gerade	2	0.029	follow	3	0.031	flight	4	0.054	bird
0	0.029	make	1	0.041	gate	2	0.026	picture	3	0.030	air	4	0.038	today
0	0.015	spot	1	0.040	hbf	2	0.026	tit	3	0.027	deutsch	4	0.031	made
0	0.013	location	1	0.030	plat	2	0.024	cute	3	0.023	fuck	4	0.020	year
5	0.288	morning	6	0.042	food	7	0.115	bahnhof	8	0.091	gehen	9	0.094	city
5	0.227	good	6	0.025	white	7	0.103	hotel	8	0.058	kommen	9	0.037	video
5	0.033	friday	6	0.021	mess	7	0.029	südkreuz	8	0.041	woche	9	0.024	trump
5	0.028	happy	6	0.019	ifs	7	0.024	gesundbrunnen	8	0.032	mehr	9	0.020	messe
5	0.026	start	6	0.017	lovely	7	0.016	lichtenstein	8	0.030	gleich	9	0.020	van
10	0.038	stag	11	0.053	immer	12	0.089	work	13	0.166	heuen	14	0.138	art
10	0.028	ram	11	0.034	party	12	0.079	job	13	0.037	stehen	14	0.106	street
10	0.028	filtern	11	0.030	berl	12	0.052	hiring	13	0.037	geben	14	0.035	graffiti
10	0.021	nofilter	11	0.016	hope	12	0.034	great	13	0.037	gestern	14	0.033	design
10	0.021	star	11	0.015	cuff	12	0.032	arc	13	0.028	danke	14	0.032	urban
15	0.035	visit	16	0.063	marathon	17	0.289	morgen	18	0.100	back	19	0.374	airport
15	0.028	olympiad	16	0.034	run	17	0.251	gut	18	0.091	time	19	0.123	tegel
15	0.021	photochemistry	16	0.033	ready	17	0.048	lieb	18	0.072	week	19	0.094	otto
15	0.021	lot	16	0.027	ostbahnhof	17	0.030	tristan	18	0.072	home	19	0.093	lilienthal
15	0.015	god	16	0.026	running	17	0.029	schönen	18	0.020	shop	19	0.051	schoolfellow
20	0.048	spandau	21	0.109	nuremberg	22	0.069	breakfast	23	0.043	fashion	24	0.049	moin
20	0.035	platz	21	0.066	park	22	0.047	face	23	0.026	myelom	24	0.039	tempelhof
20	0.026	beim	21	0.035	autobahn	22	0.028	cool	23	0.025	miss	24	0.027	scheu
20	0.024	itb	21	0.034	tor	22	0.020	super	23	0.021	style	24	0.027	open
20	0.023	birthday	21	0.020	heute	22	0.016	neural	23	0.018	dia	24	0.023	nein
25	0.061	sunrise	26	0.223	day	27	0.093	coffee	28	0.044	rot	29	0.111	wall
25	0.056	music	26	0.061	beautiful	27	0.047	place	28	0.040	klein	29	0.058	gallery
25	0.042	live	26	0.031	amazing	27	0.033	view	28	0.032	sonne	29	0.054	east
25	0.033	photography	26	0.027	long	27	0.033	office	28	0.026	ehe	29	0.053	side
25	0.030	yesterday	26	0.016	christmas	27	0.029	berg	28	0.024	laufen	29	0.027	walk
30	0.077	alexander	31	0.126	travel	32	0.064	bvg	33	0.054	start	34	0.047	sky
30	0.065	mitre	31	0.051	museum	32	0.063	kindergarten	33	0.046	great	34	0.042	fit
30	0.032	fernsehturm	31	0.049	trip	32	0.034	moringen	33	0.038	weekend	34	0.040	winter
30	0.031	girl	31	0.032	gram	32	0.033	bus	33	0.036	friend	34	0.036	summer
30	0.024	ans	31	0.027	europa	32	0.029	house	33	0.035	year	34	0.034	cloud
35	0.079	spree	36	0.075	bahn	37	0.136	hauptmann	38	0.172	flughafen	39	0.123	amp
35	0.043	architecture	36	0.071	mal	37	0.023	bye	38	0.142	tegel	39	0.038	show
35	0.040	light	36	0.054	erst	37	0.020	joy	38	0.120	rapport	39	0.037	people
35	0.030	dom	36	0.031	frühstücke	37	0.019	gehen	38	0.098	lilienthal	39	0.025	team
35	0.026	red	36	0.026	schöneweide	37	0.017	karlshorst	38	0.096	otto	39	0.021	talk
40	0.087	station	41	0.154	love	42	0.055	bar	43	0.063	today	44	0.075	memorial
40	0.048	friedrich	41	0.073	happy	42	0.028	flower	43	0.048	charlottetown	44	0.021	tower
40	0.037	train	41	0.049	heart	42	0.022	hostel	43	0.023	stratum	44	0.020	victory
40	0.026	central	41	0.034	sunday	42	0.020	panorama	43	0.022	fun	44	0.019	holocaust
40	0.020	warm	41	0.029	thing	42	0.020	lindbergh	43	0.020	flying	44	0.017	town
45	0.039	series	46	0.086	mal	47	0.089	tag	48	0.091	night	49	0.041	uhr
45	0.028	sport	46	0.067	schon	47	0.045	sehen	48	0.087	life	49	0.027	schön
45	0.026	studio	46	0.031	mussen	47	0.043	ganz	48	0.053	world	49	0.024	linie
45	0.023	woman	46	0.025	wer	47	0.029	nordhafen	48	0.040	cold	49	0.024	früh
45	0.021	workout	46	0.022	tun	47	0.026	alee	48	0.026	saturday	49	0.020	nature

Zeitschiene: 09:00 – 12:00 Uhr

0	0.064	live	1	0.133	coffee	2	0.024	toll	3	0.087	happy	4	0.097	travel
0	0.040	messe	1	0.105	love	2	0.022	repost	3	0.071	alexander	4	0.049	home
0	0.036	house	1	0.046	face	2	0.022	gross	3	0.058	year	4	0.040	trip
0	0.022	itb	1	0.038	heart	2	0.020	schönen	3	0.027	birthday	4	0.040	city
0	0.020	head	1	0.028	eye	2	0.020	sonntag	3	0.024	fernsehturm	4	0.027	gram
5	0.109	mal	6	0.075	work	7	0.054	schon	8	0.092	gehen	9	0.025	stag
5	0.098	heuen	6	0.053	job	7	0.042	geben	8	0.035	spring	9	0.020	ram
5	0.037	beim	6	0.043	cad	7	0.040	immer	8	0.033	tristan	9	0.019	arena
5	0.031	sehen	6	0.037	gay	7	0.034	kommen	8	0.020	gleich	9	0.018	mercedes
5	0.024	einfach	6	0.036	latest	7	0.031	mehr	8	0.020	endlich	9	0.018	dia
10	0.146	art	11	0.040	kurfürstendamm	12	0.086	life	13	0.121	breakfast	14	0.030	film
10	0.095	street	11	0.034	cool	12	0.042	world	13	0.112	hotel	14	0.028	super
10	0.044	autobahn	11	0.029	apple	12	0.041	people	13	0.037	uhr	14	0.021	ale
10	0.041	bvg	11	0.027	digital	12	0.028	autumn	13	0.015	steglitz	14	0.016	rain
10	0.034	urban	11	0.024	meet	12	0.024	festival	13	0.015	frühstück	14	0.014	outfit
15	0.080	park	16	0.048	make	17	0.059	week	18	0.212	morning	19	0.094	back
15	0.025	car	16	0.039	stagecoach	17	0.055	fashion	18	0.162	good	19	0.031	scheu
15	0.020	kadewe	16	0.039	rot	17	0.042	friedrich	18	0.038	sun	19	0.024	big
15	0.017	repertoire	16	0.033	bundestag	17	0.023	show	18	0.036	summer	19	0.021	flight
15	0.015	cebit	16	0.027	red	17	0.015	danke	18	0.033	monday	19	0.019	beer
20	0.090	hauptmann	21	0.070	food	22	0.083	night	23	0.093	nuremberg	24	0.042	thing
20	0.034	klein	21	0.064	great	22	0.044	club	23	0.039	tor	24	0.039	place
20	0.027	mussen	21	0.043	yesterday	22	0.040	music	23	0.033	gate	24	0.036	amazing
20	0.025	bahn	21	0.039	fun	22	0.036	party	23	0.024	link	24	0.030	favourite
20	0.022	tun	21	0.027	hour	22	0.034	friday	23	0.018	wochenende	24	0.028	uni
25	0.072	memorial	26	0.202	airport	27	0.068	tag	28	0.065	love	29	0.042	friend
25	0.043	wedding	26	0.139	tegel	27	0.029	neü	28	0.065	beautiful	29	0.041	tour
25	0.040	europa	26	0.100	lilienthal	27	0.028	series	28	0.059	sunday	29	0.034	checkpoint
25	0.022	europe	26	0.099	otto	27	0.027	letzt	28	0.037	architecture	29	0.030	charlie
25	0.020	holocaust	26	0.096	flughafen	27	0.026	woman	28	0.029	happy	29	0.019	walking
30	0.230	photo	31	0.063	station	32	0.059	deutsch	33	0.031	light	34	0.076	bar
30	0.138	posted	31	0.051	winter	32	0.022	boy	33	0.027	pretty	34	0.036	ready
30	0.027	video	31	0.037	christmas	32	0.016	challenge	33	0.021	tea	34	0.033	hallo
30	0.025	dom	31	0.036	plat	32	0.014	myelom	33	0.019	sound	34	0.026	tempelhof
30	0.021	poster	31	0.027	ice	32	0.013	bir	33	0.018	coming	34	0.026	restaurant
35	0.035	studio	36	0.047	charlottetown	37	0.044	erst	38	0.069	amp	39	0.048	centre
35	0.027	early	36	0.029	danke	37	0.036	platz	38	0.048	run	39	0.029	book
35	0.024	ifa	36	0.021	moin	37	0.029	ehe	38	0.037	marathon	39	0.025	tattoo
35	0.021	made	36	0.021	ach	37	0.028	neukölln	38	0.034	running	39	0.020	talk
35	0.020	starbucks	36	0.020	vintage	37	0.022	gestern	38	0.033	style	39	0.019	hard
40	0.098	museum	41	0.063	city	42	0.140	time	43	0.195	day	44	0.035	frühstücke
40	0.066	bahnhof	41	0.040	ifs	42	0.027	long	43	0.091	today	44	0.014	cat
40	0.029	spandau	41	0.031	mess	42	0.026	haus	43	0.067	start	44	0.011	buenos
40	0.021	berl	41	0.022	phone	42	0.025	vegan	43	0.031	sunny	44	0.011	nordhafen
40	0.016	hamburger	41	0.016	hall	42	0.023	family	43	0.022	nice	44	0.010	snap
45	0.123	wall	46	0.062	weekend	47	0.071	mitre	48	0.042	photography	49	0.147	gut
45	0.075	east	46	0.060	amp	47	0.046	office	48	0.035	kindergarten	49	0.135	morgen
45	0.072	side	46	0.032	team	47	0.040	prenuptial	48	0.031	picture	49	0.041	zoo
45	0.068	gallery	46	0.027	event	47	0.038	fit	48	0.028	colour	49	0.040	lieb
45	0.032	spree	46	0.026	palaeozaen	47	0.036	berg	48	0.027	pix	49	0.027	kaffee

Zeitschiene: 12:00 – 15:00 Uhr

0	0.052	world	1	0.040	cad	2	0.038	autobahn	3	0.071	bar	4	0.041	mehr
0	0.033	ifs	1	0.036	berg	2	0.036	bvg	3	0.023	hour	4	0.038	armut
0	0.030	stone	1	0.030	prenuptial	2	0.021	fit	3	0.023	haus	4	0.030	immer
0	0.018	ostbahnhof	1	0.021	link	2	0.020	fitness	3	0.015	head	4	0.027	jed
0	0.018	mess	1	0.020	hand	2	0.019	training	3	0.014	vocalism	4	0.022	mensch
5	0.048	place	6	0.053	face	7	0.047	checkpoint	8	0.065	work	9	0.042	kindergarten
5	0.042	cool	6	0.051	deutsch	7	0.044	vielfach	8	0.052	job	9	0.031	autumn
5	0.039	centre	6	0.033	neukölln	7	0.039	charlie	8	0.039	team	9	0.026	messe
5	0.035	people	6	0.026	hard	7	0.029	scheu	8	0.027	hiring	9	0.023	stadion
5	0.030	thing	6	0.023	star	7	0.023	kreativ	8	0.026	view	9	0.021	coming
10	0.097	memorial	11	0.111	nuremberg	12	0.053	charlottetown	13	0.161	wall	14	0.155	art
10	0.058	europa	11	0.054	friedrich	12	0.023	schloss	13	0.090	east	14	0.086	street
10	0.027	history	11	0.021	car	12	0.022	top	13	0.086	gallery	14	0.049	design
10	0.027	visit	11	0.018	baby	12	0.020	essen	13	0.080	side	14	0.041	photography
10	0.025	green	11	0.018	volkswagen	12	0.019	gendarmenmarkt	13	0.023	city	14	0.037	architecture
15	0.071	back	16	0.065	night	17	0.060	home	18	0.056	live	19	0.151	love
15	0.028	neural	16	0.045	day	17	0.025	run	18	0.052	music	19	0.085	life
15	0.026	event	16	0.037	amazing	17	0.022	day	18	0.047	house	19	0.045	heart
15	0.025	wedding	16	0.033	great	17	0.021	feel	18	0.033	store	19	0.031	beautiful
15	0.023	woman	16	0.028	yesterday	17	0.018	hallmark	18	0.029	show	19	0.030	eye
20	0.084	alexander	21	0.029	black	22	0.035	olympiad	23	0.061	gut	24	0.042	kommen
20	0.042	dom	21	0.028	red	22	0.034	bundestag	23	0.042	schon	24	0.039	alt
20	0.041	tor	21	0.021	walk	22	0.015	gor	23	0.029	neü	24	0.030	klein
20	0.036	gate	21	0.021	style	22	0.014	mall	23	0.028	mussen	24	0.027	lieb
20	0.021	fernsehturm	21	0.021	white	22	0.013	sonntag	23	0.027	morgen	24	0.021	tun
25	0.143	amp	26	0.032	beim	27	0.058	station	28	0.049	winter	29	0.046	restaurant
25	0.017	tempelhof	26	0.030	sehen	27	0.045	tour	28	0.037	christmas	29	0.040	friend
25	0.016	ifa	26	0.023	ganz	27	0.041	studio	28	0.031	plat	29	0.029	good
25	0.015	woche	26	0.022	einfach	27	0.019	repost	28	0.022	market	29	0.027	office
25	0.015	wochenende	26	0.022	meet	27	0.017	train	28	0.022	pix	29	0.023	family
30	0.039	kurfürstendamm	31	0.125	coffee	32	0.048	club	33	0.092	today	34	0.084	travel
30	0.036	burger	31	0.054	morning	32	0.043	zoo	33	0.045	week	34	0.042	mitre
30	0.030	stag	31	0.053	good	32	0.022	sound	33	0.034	spring	34	0.030	ale
30	0.028	apple	31	0.025	start	32	0.019	cloud	33	0.030	time	34	0.030	trip
30	0.025	stagecoach	31	0.023	break	32	0.019	official	33	0.029	book	34	0.026	holiday
35	0.028	made	36	0.054	platz	37	0.067	gehen	38	0.025	air	39	0.186	airport
35	0.027	light	36	0.023	kind	37	0.066	hauptmann	38	0.021	arena	39	0.115	tegel
35	0.024	stop	36	0.022	sky	37	0.042	vegan	38	0.018	lounge	39	0.079	otto
35	0.020	mitte	36	0.020	blue	37	0.025	hallo	38	0.018	mercedes	39	0.076	lilienthal
35	0.019	picture	36	0.017	sunset	37	0.024	gleich	38	0.016	flight	39	0.066	flughafen
40	0.121	museum	41	0.042	kadewe	42	0.088	park	43	0.238	photo	44	0.086	food
40	0.064	bahnhof	41	0.019	thursday	42	0.030	shop	43	0.159	posted	44	0.072	lunch
40	0.040	fashion	41	0.016	victory	42	0.028	spree	43	0.034	video	44	0.035	breakfast
40	0.026	hamburger	41	0.015	tuesday	42	0.019	style	43	0.028	make	44	0.024	porn
40	0.016	kaffee	41	0.013	data	42	0.019	berl	43	0.024	post	44	0.024	itb
45	0.041	beer	46	0.115	day	47	0.086	happy	48	0.105	time	49	0.095	mal
45	0.034	spandau	46	0.051	sunday	47	0.078	hotel	48	0.063	year	49	0.076	heuen
45	0.032	tag	46	0.045	summer	47	0.048	party	48	0.030	film	49	0.031	erst
45	0.031	ice	46	0.039	sun	47	0.040	city	48	0.028	favourite	49	0.024	danke
45	0.030	rot	46	0.038	weekend	47	0.033	birthday	48	0.023	long	49	0.019	sagen

Zeitschiene: 15:00 – 18:00 Uhr

0	0.026	bit	1	0.033	alt	2	0.055	alexanderplatz	3	0.040	world	4	0.084	back
0	0.020	future	1	0.031	bundestag	2	0.028	book	3	0.026	weekend	4	0.069	home
0	0.017	project	1	0.028	super	2	0.026	light	3	0.023	long	4	0.039	week
0	0.015	straße	1	0.024	sky	2	0.020	picture	3	0.022	saturday	4	0.030	tempelhof
0	0.012	head	1	0.021	steglitz	2	0.018	tattoo	3	0.021	walk	4	0.018	baby
5	0.038	people	6	0.025	start	7	0.122	art	8	0.053	beer	9	0.242	photo
5	0.025	green	6	0.023	sollen	7	0.045	street	8	0.038	deutsch	9	0.162	posted
5	0.022	red	6	0.023	neü	7	0.035	design	8	0.029	burger	9	0.059	drinking
5	0.021	blue	6	0.022	zwei	7	0.029	architecture	8	0.029	arena	9	0.030	dom
5	0.020	fernsehturm	6	0.019	mitre	7	0.028	graffiti	8	0.027	mercedes	9	0.016	ipa
10	0.054	charlie	11	0.065	txl	12	0.146	love	13	0.185	airport	14	0.046	kurfürstendamm
10	0.052	checkpoint	11	0.050	air	12	0.054	sunday	13	0.117	tegel	14	0.033	apple
10	0.034	hauptbahnhof	11	0.050	flugzeugbildde	12	0.037	heart	13	0.079	flughafen	14	0.031	store
10	0.029	ice	11	0.049	airbus	12	0.020	girl	13	0.076	otto	14	0.018	tempelhofer
10	0.027	check	11	0.023	friedrichshain	12	0.018	fun	13	0.074	lilienthal	14	0.015	feld
15	0.066	food	16	0.043	tor	17	0.065	great	18	0.028	berg	19	0.033	bvg
15	0.047	stone	16	0.031	gate	17	0.061	work	18	0.023	endlich	19	0.032	bahn
15	0.023	restaurant	16	0.020	black	17	0.043	job	18	0.022	itb	19	0.031	neukölln
15	0.023	chocolate	16	0.017	monday	17	0.031	team	18	0.021	ehe	19	0.021	autobahn
15	0.020	lunch	16	0.014	making	17	0.026	hiring	18	0.015	prenzlauer	19	0.017	shot
20	0.129	museum	21	0.144	day	22	0.028	video	23	0.055	mitte	24	0.046	station
20	0.034	cool	21	0.073	today	22	0.021	exhibition	23	0.043	place	24	0.045	alexander
20	0.030	charlottenburg	21	0.039	nice	22	0.015	temperate	23	0.024	essen	24	0.044	hauptmann
20	0.025	rot	21	0.035	sun	22	0.014	doe	23	0.021	center	24	0.021	geht
20	0.024	hallo	21	0.027	visit	22	0.012	special	23	0.017	rock	24	0.019	messe
25	0.068	platz	26	0.065	mehr	27	0.102	memorial	28	0.050	nuremberg	29	0.065	bar
25	0.036	beim	26	0.065	immer	27	0.046	europe	28	0.047	winter	29	0.060	life
25	0.019	workout	26	0.030	house	27	0.028	murdered	28	0.033	made	29	0.043	zoo
25	0.019	ebb	26	0.020	mensch	27	0.026	europa	28	0.025	snow	29	0.021	stop
25	0.017	klein	26	0.016	jahr	27	0.024	jews	28	0.022	gesund	29	0.020	view
30	0.160	wall	31	0.083	hotel	32	0.064	armut	33	0.045	kadewe	34	0.027	thing
30	0.088	east	31	0.025	sunset	32	0.028	kinder	33	0.036	vielfach	34	0.024	caf
30	0.078	side	31	0.021	open	32	0.021	leben	33	0.026	kaufhaus	34	0.021	favourite
30	0.078	gallery	31	0.019	pizza	32	0.019	seit	33	0.024	uhr	34	0.021	ifa
30	0.015	west	31	0.018	hour	32	0.017	kind	33	0.020	shopping	34	0.019	car
35	0.037	olympiastadion	36	0.071	bahnhof	37	0.085	mal	38	0.138	amp	39	0.041	spandau
35	0.026	building	36	0.027	danke	37	0.045	heuen	38	0.032	ale	39	0.035	show
35	0.024	reichstag	36	0.025	hamburger	37	0.037	heute	38	0.022	part	39	0.029	fashion
35	0.019	top	36	0.016	rfc	37	0.027	gut	38	0.018	film	39	0.022	festival
35	0.017	walking	36	0.013	dog	37	0.026	erst	38	0.016	friends	39	0.019	amp
40	0.029	marketeer	41	0.084	park	42	0.051	year	43	0.078	travel	44	0.027	berl
40	0.019	found	41	0.056	summer	42	0.044	christmas	43	0.076	city	44	0.019	berlim
40	0.019	style	41	0.038	beautiful	42	0.025	family	43	0.065	kreuzberg	44	0.015	alemania
40	0.018	sport	41	0.030	autumn	42	0.019	market	43	0.023	flag	44	0.014	gross
40	0.018	fitness	41	0.027	spree	42	0.017	ikea	43	0.020	photography	44	0.013	siegessäule
45	0.050	schon	46	0.068	happy	47	0.037	geben	48	0.123	time	49	0.055	live
45	0.025	wer	46	0.049	night	47	0.035	spring	48	0.077	coffee	49	0.044	vegan
45	0.021	zeit	46	0.040	music	47	0.031	gehen	48	0.073	good	49	0.044	amazing
45	0.019	lieb	46	0.038	party	47	0.027	marathon	48	0.021	shop	49	0.029	studio
45	0.018	gut	46	0.038	club	47	0.024	wedding	48	0.018	friend	49	0.019	lounge

Zeitschiene: 18:00 – 21:00 Uhr

0	0.054	zoo	1	0.033	arena	2	0.033	schon	3	0.020	casa	4	0.062	food
0	0.051	friends	1	0.027	mercedes	2	0.022	ganz	3	0.010	hoy	4	0.055	dinner
0	0.029	fun	1	0.025	benz	2	0.022	jahr	3	0.010	smoke	4	0.023	pizza
0	0.029	cafe	1	0.025	beim	2	0.018	wer	3	0.009	che	4	0.022	foodporn
0	0.025	selfie	1	0.021	festival	2	0.017	gleich	3	0.008	solo	4	0.021	pic
5	0.044	abend	6	0.060	tor	7	0.057	happy	8	0.059	great	9	0.075	memorial
5	0.034	house	6	0.049	charlottenburg	7	0.036	straße	8	0.049	weekend	9	0.053	europe
5	0.026	igers	6	0.044	reichstag	7	0.028	haus	8	0.038	nice	9	0.043	dom
5	0.019	open	6	0.041	berl	7	0.027	birthday	8	0.036	place	9	0.026	tattoo
5	0.015	musik	6	0.038	gate	7	0.026	vegan	8	0.023	bit	9	0.018	murdered
10	0.202	amp	11	0.049	mehr	12	0.041	olympiastadion	13	0.091	today	14	0.064	park
10	0.038	friedrichshain	11	0.036	immer	12	0.022	berlim	13	0.030	spree	14	0.023	tempelhofer
10	0.027	caf	11	0.035	neue	12	0.013	hertha	13	0.018	red	14	0.020	feld
10	0.022	uci	11	0.033	menschen	12	0.012	foto	13	0.017	chocolate	14	0.019	life
10	0.017	vsco	11	0.018	viele	12	0.012	steglitz	13	0.017	cinestar	14	0.016	treptower
15	0.040	people	16	0.177	airport	17	0.087	art	18	0.093	mal	19	0.031	bvg
15	0.035	beautiful	16	0.097	tegel	17	0.056	bahnhof	18	0.081	heute	19	0.031	bahn
15	0.032	life	16	0.066	flughafen	17	0.032	streetart	18	0.021	endlich	19	0.029	ubahn
15	0.031	coffee	16	0.061	otto	17	0.026	graffiti	18	0.017	schon	19	0.022	liebe
15	0.029	amazing	16	0.059	lilienthal	17	0.026	berg	18	0.015	morgen	19	0.016	bus
20	0.051	christmas	21	0.078	live	22	0.039	sunday	23	0.253	photo	24	0.095	bar
20	0.024	studio	21	0.047	music	22	0.034	architecture	23	0.195	posted	24	0.062	restaurant
20	0.021	check	21	0.039	konzert	22	0.030	winter	23	0.034	video	24	0.031	schu
20	0.020	big	21	0.033	show	22	0.026	evening	23	0.010	camera	24	0.019	berghain
20	0.018	meetup	21	0.024	theater	22	0.023	cold	23	0.009	heinersdorf	24	0.014	panorama
25	0.090	drinking	26	0.097	mitte	27	0.121	day	28	0.101	time	29	0.157	love
25	0.054	beer	26	0.059	neukölln	27	0.026	wedding	28	0.031	days	29	0.089	night
25	0.036	stone	26	0.022	black	27	0.019	green	28	0.027	finally	29	0.043	heart
25	0.034	ale	26	0.015	neukoelln	27	0.018	bad	28	0.026	years	29	0.026	eyes
25	0.027	photo	26	0.014	dog	27	0.016	lovely	28	0.023	long	29	0.016	light
30	0.067	home	31	0.016	monday	32	0.033	fashion	33	0.050	hauptbahnhof	34	0.028	spandau
30	0.030	kino	31	0.015	thursday	32	0.029	make	33	0.042	station	34	0.019	smile
30	0.023	autumn	31	0.014	bundestag	32	0.022	style	33	0.041	center	34	0.018	happy
30	0.021	iphone	31	0.012	year	32	0.022	model	33	0.029	sony	34	0.017	rathaus
30	0.017	marathon	31	0.011	police	32	0.019	photography	33	0.016	ice	34	0.017	shopping
35	0.042	gibt	36	0.036	mauerpark	37	0.103	museum	38	0.069	good	39	0.098	platz
35	0.031	danke	36	0.032	spring	37	0.024	design	38	0.031	flag	39	0.031	geht
35	0.030	gut	36	0.029	deutsche	37	0.016	jewish	38	0.023	garten	39	0.021	fcr
35	0.018	schön	36	0.026	film	37	0.014	opening	38	0.016	stop	39	0.020	stadion
35	0.018	wochenende	36	0.025	tag	37	0.014	exhibition	38	0.016	joy	39	0.012	analten
40	0.128	kreuzberg	41	0.047	sunset	42	0.047	work	43	0.089	alexanderplatz	44	0.060	back
40	0.074	hotel	41	0.026	nofilter	42	0.041	tiergarten	43	0.050	summer	44	0.036	cool
40	0.018	bikini	41	0.021	ifa	42	0.029	team	43	0.039	kurfürstendamm	44	0.035	kadewe
40	0.016	room	41	0.020	messe	42	0.015	fire	43	0.027	fernsehturm	44	0.021	super
40	0.014	hours	41	0.019	week	42	0.015	training	43	0.027	sun	44	0.015	kaufhaus
45	0.136	wall	46	0.057	armut	47	0.056	burger	48	0.071	city	49	0.069	party
45	0.094	east	46	0.026	weihnachtsmarkt	47	0.028	essen	48	0.059	travel	49	0.058	club
45	0.085	side	46	0.024	gendarmenmarkt	47	0.021	top	48	0.039	world	49	0.038	tonight
45	0.081	gallery	46	0.014	pankow	47	0.012	beef	48	0.037	checkpoint	49	0.032	flugzeugbildde
45	0.019	west	46	0.013	kaiser	47	0.011	pink	48	0.037	charlie	49	0.032	txl

Zeitschiene: 21:00 – 24:00 Uhr

0	0.024	lollapalooza	1	0.136	wall	2	0.057	berl	3	0.097	hotel	4	0.046	schon
0	0.015	thursday	1	0.087	east	2	0.021	alemania	3	0.027	igers	4	0.040	danke
0	0.015	trump	1	0.078	side	2	0.015	vsco	3	0.019	hostel	4	0.039	geht
0	0.014	left	1	0.078	gallery	2	0.015	instagram	3	0.018	autumn	4	0.031	gut
0	0.012	meeting	1	0.038	gate	2	0.011	hoy	3	0.018	marathon	4	0.021	morgen
5	0.038	sunset	6	0.107	day	7	0.186	amp	8	0.053	bahnhof	9	0.056	hauptbahnhof
5	0.034	fernsehturm	6	0.041	year	7	0.034	burger	8	0.031	berlim	9	0.037	summer
5	0.032	visit	6	0.038	beautiful	7	0.020	miss	8	0.019	hamburger	9	0.024	olympiastadion
5	0.027	view	6	0.038	winter	7	0.018	run	8	0.012	foto	9	0.024	spandau
5	0.025	teufelsberg	6	0.023	coming	7	0.017	black	8	0.011	ramen	9	0.021	place
10	0.175	love	11	0.073	tor	12	0.019	einfach	13	0.238	photo	14	0.111	time
10	0.048	heart	11	0.038	bit	12	0.015	series	13	0.189	posted	14	0.060	amazing
10	0.026	selfie	11	0.037	station	12	0.015	schön	13	0.032	video	14	0.024	long
10	0.022	girl	11	0.025	bahn	12	0.014	zwei	13	0.024	stone	14	0.022	things
10	0.022	eyes	11	0.021	real	12	0.014	frauen	13	0.008	camera	14	0.021	finally
15	0.071	live	16	0.044	dom	17	0.043	weekend	18	0.148	night	19	0.036	armut
15	0.060	music	16	0.022	team	17	0.032	tag	18	0.091	good	19	0.032	zoo
15	0.039	open	16	0.019	wonderful	17	0.031	nacht	18	0.035	evening	19	0.020	leben
15	0.022	mic	16	0.019	wine	17	0.029	gute	18	0.024	saturday	19	0.020	macht
15	0.022	concert	16	0.016	feeling	17	0.017	style	18	0.017	tempelhofer	19	0.018	kinder
20	0.144	bar	21	0.038	immer	22	0.081	party	23	0.047	world	24	0.029	fast
20	0.057	park	21	0.033	charlie	22	0.031	years	23	0.018	baby	24	0.026	liebe
20	0.039	tiergarten	21	0.033	checkpoint	22	0.023	ready	23	0.016	vielfach	24	0.026	light
20	0.030	cafe	21	0.031	menschen	22	0.020	ago	23	0.012	magic	24	0.025	theater
20	0.027	berghain	21	0.029	mehr	22	0.014	dark	23	0.012	dog	24	0.020	columbia
25	0.030	arena	26	0.116	kreuzberg	27	0.055	neukölln	28	0.058	food	29	0.154	airport
25	0.026	gendarmenmarkt	26	0.081	art	27	0.042	photography	28	0.049	restaurant	29	0.074	tegel
25	0.023	tour	26	0.045	streetart	27	0.016	myleo	28	0.048	dinner	29	0.053	flughafen
25	0.022	mercedes	26	0.027	street	27	0.013	neukoelln	28	0.025	house	29	0.048	otto
25	0.021	benz	26	0.026	graffiti	27	0.012	lido	28	0.019	foodporn	29	0.048	schönefeld
30	0.088	mitte	31	0.080	home	32	0.025	made	33	0.077	memorial	34	0.088	great
30	0.025	drinks	31	0.047	work	32	0.022	casa	33	0.071	travel	34	0.046	show
30	0.019	drink	31	0.041	back	32	0.019	post	33	0.058	reichstag	34	0.034	tonight
30	0.014	garden	31	0.027	life	32	0.018	make	33	0.057	europe	34	0.026	awesome
30	0.013	zob	31	0.023	red	32	0.016	favorite	33	0.041	flag	34	0.022	studio
35	0.067	happy	36	0.119	drinking	37	0.040	neue	38	0.038	straße	39	0.099	platz
35	0.032	birthday	36	0.045	ale	37	0.033	kino	38	0.027	tomorrow	39	0.040	center
35	0.015	joy	36	0.042	photo	37	0.026	film	38	0.026	back	39	0.032	sony
35	0.014	palast	36	0.018	pub	37	0.023	fcr	38	0.022	friday	39	0.019	original
35	0.014	astra	36	0.015	brauerei	37	0.021	welt	38	0.016	throwback	39	0.013	cheers
40	0.054	club	41	0.037	kurfürstendamm	42	0.055	charlottenburg	43	0.028	gerade	44	0.086	mal
40	0.053	friends	41	0.029	mauerpark	42	0.028	today	43	0.017	running	44	0.064	heute
40	0.027	people	41	0.026	fashion	42	0.021	weihnachtsmarkt	43	0.016	store	44	0.053	christmas
40	0.025	sunday	41	0.020	hope	42	0.019	palace	43	0.014	finished	44	0.039	abend
40	0.024	cool	41	0.020	haus	42	0.018	pizza	43	0.014	opening	44	0.033	beim
45	0.093	city	46	0.042	today	47	0.078	museum	48	0.097	alexanderplatz	49	0.059	beer
45	0.035	friedrichshain	46	0.036	nice	47	0.026	spring	48	0.031	berg	49	0.017	big
45	0.030	super	46	0.031	fun	47	0.025	architecture	48	0.029	spree	49	0.017	free
45	0.025	lights	46	0.024	pic	47	0.020	church	48	0.026	prenzlauer	49	0.017	week
45	0.022	coffee	46	0.021	instagood	47	0.018	kadewe	48	0.025	urban	49	0.016	itb