NSA und Co - Auswertung von Massendaten in *R* am Beispiel E-Mail-Korrespondenz

Ein Beitrag von Clemens von Bismarck-Osten, Data Scientist bei OPITZ CONSULTING

Einleitung

Seit den Enthüllungen der NSA-Aktivitäten durch ehemalige Mitarbeiter (sogenannte „Whistleblower) rückt die Thematik der generellen und verdachtsunabhängigen Überwachung auch bei uns in Deutschland immer weiter in den Vordergrund. Dies ist vor allem der Tatsache geschuldet, dass eigentlich alle modernen Kommunikationswege davon betroffen waren und sind, egal ob es sich um Telefonate, Textmitteilungen oder E-Mails handelt.
Ausgewertet wird dabei: der eigentliche Inhalt, Aufenthaltsstandort(e) und damit Bewegungsprofile, Zeitpunkt, benutzte Hardwarekomponenten, und viele weitere. Diese Daten werden auf bekannte Muster untersucht, oder sie dienen dazu, neue Muster in den Daten zu identifizieren und die vorhandenen Erkennungs-Algorithmen zu optimieren.

Aber was genau lässt sich eigentlich auswerten und ab wann mache ich mich als User „verdächtig“, gerate also in den engeren Kreis potenziell Verdächtiger? Mit absoluter Sicherheit lässt sich dies ohne konkreten Einblick in die Praxis wohl nicht präzise sagen. Dennoch braucht es kein Detailwissen, um zu mutmaßen, dass ein bestimmtes Nutzerverhalten wie das Verschicken von Texten, die bestimmte Schlagwörter enthalten, die Chance erhöht, in die engere Auswahl „verdächtiger Elemente“ zu gelangen.

Im Folgenden sollen deswegen ein paar recht einfache Auswertungsschritte in *R* aufgezeigt werden, wie man bei der Analyse von Texten vorgehen kann. Die fachliche Tiefe beruht bei der gezeigten Thematik eher auf Vermutungen und „*Common Sense*“ und erhebt keinesfalls den Anspruch, die Praxis von Geheimdiensten und Spähsoftware widerzuspiegeln. Vielmehr zielt das gezeigte Beispiel darauf, zu zeigen, wie Text Mining in R abläuft und welche Ansätze man für einfache Analysen wählen kann.

Methoden

Grundlage der Auswertungen – Erstellen eines Textcorpus

Die Grundlage für alle Auswertungen sollte eine genügend große Anzahl an Texten darstellen. Für eine aussagekräftige und statistisch aussagekräftige Textklassifikation empfiehlt sich in diesem Fall eine Dokumentenanzahl von n > 10.000. Für eine möglichste heterogene Datenbasis sollten - wenn möglich - Texte unterschiedlichster Herkunft (eMails, SMS, Beiträge uvm.) enthalten sein, um eine möglichst große Bandbreite an Textgrößen und –inhalten zu schaffen.

Die in diesem Beispiel verwendete Auswertung-Grundlage beinhaltet 50.000 Datensätze und stammt aus öffentlich zugänglichen Quellen, die dennoch hier rein exemplarisch

Verwendung finden und aus datenschutzrechtlichen Gründen keine Klarnamen, URLs oder E-Mailadressen enthalten.

Die Roh-Daten (hier in ‚*g*') sollten dabei zunächst in auswertbaren Blöcken (bspw. 200 MB) in der Datenbank abgelegt werden. Erst zur Auswertung werden sie in ein einheitliches Format überführt. Wie diese Einheitlichkeit aussieht, sollte dabei im Vorfeld festgelegt werden. Für hierarchisch geschachtelte Informationen empfiehlt sich eine XML- Struktur o.ä., für unser Beispiel reicht eine einfache Tabellenstruktur, wie in Tab. 1 gezeigt.

Folgender Code-Abschnitt liest Dokumente hinter einer URL, die fortlaufend nummeriert sind, parallelisiert aus und legt den Inhalt in einem data.frame ‚*g*' mit 4 Spalten ab (*'Time', 'From', 'To', 'Message'*). Dafür muss eine Anpassung der Zeilen 33 und 54 bis 59 vorgenommen werden.

```
1
2
3   # java Home Pfad anpassen
4   Sys.setenv(JAVA HOME="C:/Program Files/Java/jdk1.7.0 71")
5
6   # load packages
7   library(RCurl)
8   library(XML)
9   library(stringr)
10  library(stringi)
11  library(sqldf)
12  library(tm)
13  library(class)
14  library(rJava)
15  library(RWeka)
16  library(wordcloud)
17  library(plyr)
18  library(akmeans)
19  library(caret)
20  library(doParallel)
21
22  # https auslesen
23  setInternet2(TRUE)
24  options(stringsAsFactors = FALSE)
25
26  # Inhalt einer URL mit parallelen Sessions auslesen
27  detectCores()
28  registerDoParallel(cores=8)
29  getDoParWorkers()
30
31  # findet ein enthaltendes Datum im Standard Format ‚YYYY-MM-DD hh:mm:ss'
32  regex date = '[0-9]{4}-[0-9]{2}-[0-9]{2} [0-9]{2}:[0-9]{2}:[0-9]{2}'
33  path1 = ('https://***/files/emailid/')
34
35  size = 50000
36  b = sample(1:5000000, size, replace=FALSE)
37  a = rep(path1, each = size, len = size)
38  c = paste0(a, b)
39
40
41  corpi   =   foreach(i=c,   .combine=rbind,   .inorder=FALSE,   .packages=c('RCurl',   'stringr'),
42  .errorhandling=c('remove')) %dopar%
43      {
44
45      raw = lapply(i, getURL)
46      pages = lapply(raw, iconv, from="WINDOWS-1252", to="UTF-8")
47
48      pages2 = as.character(pages)
49      pages2 = matrix(unlist(pages2), byrow = TRUE)
50
51      c1 = str extract(pages2, regex date)
52
53  # Die Elemente c2, c3 und c4 werden hier über die str locate funktion gefüllt
54      c2   =   substr(pages2,   str locate(pages2,   ‚START ZEICHENFOLGE'),   str locate(pages2,
55  'ENDE ZEICHENFOLGE'))
56      c3 = substr(pages2, str locate(pages2, 'START ZEICHENFOLGE'), str locate(pages2,
57  'ENDE ZEICHENFOLGE'))
58      c4   =   substr(pages2,   str locate(pages2,   'START ZEICHENFOLGE'),   str locate(pages2,
59  'ENDE ZEICHENFOLGE'))
60
61      return(data.frame(c1, c2, c3, c4))
62
63      }
64
65  # Ergebnis in Variable ‚g' schreiben und Spalten benennen
```

```
g = corpi
colnames(g) = c('Time', 'From', 'To', 'Message')
```

Tabelle 1: Inhalt von data.frame g

Parameter	Format	Beispielcontent
Zeit	*date*	*01.04.2012 07:11:03*
Absender	*char*	*max.mustermann@gmail.com*
Empfänger	*char*	*monika.musterfrau@hotmail.com*
Nachricht	*char*	*„Hallo Schatz,wie geht es Dir ?“*

Die Datenaufbereitung

Will man mit den Daten arbeiten, so müssen zunächst eine Reihe an Bereinigungs- und Modifikationsroutinen durchgeführt werden. So werden etwa für einfache Text Mining Analysen alle unwichtigen Bestandteile, wie Satzzeichen und Sonderzeichen aus dem Text entfernt, alle Großbuchstaben durch jeweils kleine ersetzt und einzeln stehende Zahlen entfernt.

1. Schritt: doppelte Nachrichten entfernen (z.B. Spam Mails, die tausendfach verschickt werden oder automatische System-Messages, wie Abwesenheitsnotizen)

```
g = g[ !duplicated(g$Message), ]
```

2. Schritt: Großbuchstaben ersetzen

```
g$Message = stri trans tolower(g$Message, locale = NULL)
```

3. Schritt: Zeilenumbrüche entfernen

```
g$Message = gsub('\n', ' ', g$Message)
```

4. Schritt: # HTML-Tags entfernen

```
g$Message = gsub("<\\/?\\w[^>]*>", " ", g$Message, perl=TRUE)
```

5. Schritt: URLs herausziehen und anschließend im Text durch „URL“ ersetzen

```
g$mentionedURL = sapply( g$Message,
                         function(x)
                        {
                          ww     =   regmatches(x,   gregexpr('http[s]?://(?:[a-zA-Z]|[0-9]|[$-
 @.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', x))
                          ww2 = do.call(paste, c(as.list(ww)))
                          ww3 = paste(ww2, collapse = ' ')
                          return(ww3)
                        }
)

g$Message = gsub('http[s]?://(?:[a-zA-Z]|[0-9]|[$- @.&+]|[!*\\(\\),]|(?:%[0-9a-fA-F] [0-9a-fA-
F]))+', 'URL', g$Message)
```

6. Schritt: E-Mail Adressen herausziehen und anschließend durch "email" ersetzen

```
g$mentionedMail = sapply( g$Message,
                                function(x)
                                {
                                  ww    =  regmatches(x,  gregexpr("\\b[a-z0-9. %+-]+@[a-z0-9.-
]+\\.[a-z]{2,4}\\b", x))
                                  ww2 = do.call(paste, c(as.list(ww)))
                                  ww3 = paste(ww2, collapse = ' ')
                                  return(ww3)
                                }
       )

g$Message  =  gsub("\\b[a-z0-9._%+-]+@[a-z0-9.-]+\\.[a-z]{2,4}\\b",  "emailaddress",  g$Message,
perl=T)
       )
```

7. Schritt: Sonderzeichen, zusätzliche Leerzeichen und einzeln stehende Zahlen entfernen

```
g$Message = gsub("[^a-zA-Z0-9 'öäüßÖÄÜ-]", " ", g$Message)

g$Message = gsub("(?<=[\\s])\\s*|^\\s+$", "", g$Message, perl=TRUE)

g$Message = gsub('\\s*(?<!\\B|-)\\d+(?!\\B|-)\\s*', " ", g$Message, perl=TRUE)
```

Die Scores

Aus den bereinigten Texten in *g$Message* lassen sich nun weitere relevante Informationen herausziehen oder errechnen, die wir hier im Folgenden „Scores“ nennen. So kann beispielsweise jede Nachricht auf ihre Übereinstimmung mit einer sogenannten „Selektorenliste“ überprüft werden, eine Liste, die beispielsweise Begriffe, Namen und Schlagwörter enthält, die im Zusammenhang mit bestimmten Themen (siehe aktuell islamistische Terrorabwehr) eine Rolle spielen. Die hier benutzte Beispiel-Liste enthält Begriffe wie:

Account, Acid, Advance, Aerial, Agency, Aggression, Aggressor, Agitator, Aim, Aircraft, Airfield, Airplane, Airport, Alarm, Alert, Alliance, Allies, Ambush, Ammunition, Anarchy, Anguish, Annihilate, Apartheid, Appeasement, Armament, Armed forces, Armory, Arms, Arsenal, Artillery, Ashore, Assassin, Assassinate, Assault und viele weitere.

```
# Selektorenliste laden und aufbereiten
Selektorenliste = scan('D:/Selektoren.txt', what='character', comment.char=';')

Selektorenliste = gsub(",", "", Selektorenliste)

Selektorenliste = stri trans tolower(Selektorenliste, locale = NULL)
```

Am Ende steht ein dimensionsloser Wert, der umso höher ist, je größer die Trefferquote mit der sogenannten Selektorenliste. Es folgt der *R* Code zur Errechnung von drei unterschiedlichen Scores.

Score 1 ist die Zahl der (unique) Wörter, die im Text gefunden wurden und gleichzeitig unter den Selektoren sind.

Score 2 ist die Zahl der (unique) Wörter im Quadrat, die im Text gefunden wurden und gleichzeitig unter den Selektoren sind, dividiert durch die Anzahl an Wörtern im Text.

Score 3 ist die Zahl der (unique) Wörter Wörter im Quadrat, die im Text gefunden wurden und gleichzeitig unter den Selektoren sind, dividiert durch die logarithmierte Anzahl an Wörtern im Text.

Der folgende Code-Abschnitt ermittelt über eine Funktion die Scores und schreibt die Ergebnisse in *g$Scores, g$Scores2, g$Scores3* und die häufigsten Wörter daraus in das Element *g$frequentWords.*

```
# Funktion zum Errechnen der Scores und der am häufigsten gefundenen Wörter
score.sentiment = function(sentences, selektorenListe, .progress='none')

{
  scores = laply(sentences, function(sentence, selektorenListe) {

    # split into words
    word.list = str split(sentence, '\\s+')

    words = unlist(word.list)

    pos.matches = match(words, selektorenListe)
    pos.matches temp = pos.matches[which(!is.na(pos.matches))]
    pos.matches = length(unique(pos.matches temp))

    #Häufigste Begriffe
    frequentWords = Selektorenliste[pos.matches temp]
    frequentWords = sort(table(frequentWords), decreasing=TRUE)
    frequentWords = names(frequentWords)

    if (length(frequentWords) >= 4) {
      frequentWords = paste(frequentWords[1:4], collapse='|')
    }
    else {
      frequentWords = paste(frequentWords, collapse='|')
    }

    score = pos.matches
    score2 = pos.matches^2 / length(words)
    score3 = pos.matches^2 / log10(length(words))

    return(c(score, score2, score3, frequentWords))

  }, selektorenListe, .progress=.progress )


scores.df   =   data.frame(score=scores[   ,1],   score2=scores[   ,2],   score3=scores[   ,3],
frequentWords=scores[ ,4])

  return(scores.df)
}

# Funktionsaufruf
scores = score.sentiment(g$Message, Selektorenliste)
g$Scores =  as.numeric(scores[ ,1])
g$Scores2 = as.numeric(scores[ ,2])
g$Scores3 = as.numeric(scores[ ,3])
g$frequentWords = as.character(scores[ ,4])

g = g[!is.na(g$Scores), ]

# Sichern
write.table(g, "c:/Daten.csv", sep = ";", row.names = FALSE)
```

Die Locations

Möchte man die Texte um zusätzliche Orts-Attribute anreichern, so kann man den Text nach Elementen einer Ortsliste (enthalten sind weltweit alle Städte über 20.000 Einwohner) und einer Länderliste durchsuchen.

```
# Cities laden
countryList = read.csv(''D:/All Cities WorldWide.csv', sep=';', header = TRUE, dec='.')
countryList = subset(countryList, nchar(as.character(City)) > 3)
countryList = subset(countryList, Population > 100000)

# Countries laden
countryList2 = read.csv('D:/countries.csv', sep=',', header = TRUE, dec='.')
countryList2 = tolower(countryList2$English.Name)
countryList2 = gsub("[^a-zA-Z0-9 'öäüßÖÄÜ]", " ", countryList2)
```

```
suchVector = as.vector(c(as.character(countryList$City), as.character(countryList2)))
Text = g$Message

# Städte mit Doppel- und Dreifachnamen einzeln durchsuchen
suchVectorTokenized = gsub(" ", "@", suchVector)
indexDoppel = which(grepl("@", suchVectorTokenized))
nurDoppelnamen = suchVectorTokenized[indexDoppel]


for (i in 1:length(nurDoppelnamen)) {
  Text = gsub(suchVector[indexDoppel][i], nurDoppelnamen[i], Text)
  print(paste0(i*length(Text), ` Dokumente durchsucht!’))
}

# Funktion
findCity = function(sentences, countryList)
{
  scores = laply(sentences, function(sentence, countryList2)
  {
    word.list = str split(sentence, '\\s+')
    words = unlist(word.list)

    pos.matches = match(words, countryList2)
    pos.matches = as.factor(pos.matches[!is.na(pos.matches)])

    best = names(sort(summary(pos.matches), decreasing=TRUE))[1:3]
    if (is.null(best) || is.na(best))
    {
      return (c(NA, NA, NA))
    }

    countryIndex = as.integer(best)
    return(countryList2[countryIndex])

  }, countryList)

  country.df = scores
  return(country.df)
}

# Funktionsaufruf
Countries = findCity(Text, suchVectorTokenized)
Countries = gsub('@', ' ', Countries)

g$City1 = as.factor(Countries[ ,1])
g$City2 = as.factor(Countries[ ,2])
g$City3 = as.factor(Countries[ ,3])
```

Die Document Term Matrix

Für eine mathematisch-statistische Auswertung wird der Text zunächst in die sogenannte Dokument-Term Matrix (DTM) überführt, d.h. jedes Wort, welches sich in allen Dokumenten mindestens so oft auftaucht, wie man gesondert definiert hat (beispielsweise in mindestens 1 % aller Dokumente), wird aufgelistet und für jeden Text gezählt. Anschließend wird die Häufigkeit über ein erweitertes TF-IDF Verfahren normiert (Siehe Abb. 1), d.h. man nimmt die Anzahl des Wortes pro Dokument und multipliziert sie mit dem logarithmierten Quotienten aus Anzahl Dokumente und der Auftretens-Häufigkeit des Wortes in allen Dokumenten. So werden häufig vorkommende Wörter weit niedriger gewichtet als jene, die

selten auftauchen und damit wahrscheinlich charakteristisch für die Inhalte eines Textes sind.

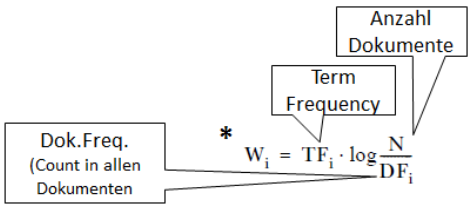


Abbildung 1: TF-IDF Normierung

Vor dem eigentlichen Zählen der Wörter werden diese noch sprachenabhängig auf ihren Wortstamm reduziert. Aus „geht“ oder „ging“ wird also „geh“, wodurch sehr viel mehr Treffer pro Einzelwort möglich sind und ein Datenrauschen vermindert wird.

Die Zerlegung in die DTM und die weitere Verarbeitung inklusive Wordcloud-Generierung sind im folgenden Block aufgeführt:

```
cleanText = g$Message
length(cleanText)
text corpus nongerman <- Corpus(VectorSource(cleanText))

# hier können von Hand zusätzliche Wörter wie einzeln auftretende Buchstaben entfernt werden
stopwords2 = c( 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l',
                'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'Wort1')

# Stemming
c1 <- tm_map(text_corpus_nongerman, removeWords, c(stopwords("english"), stopwords2))
#c1$content[[1]][[1]]

c1 <- tm_map(c1, stripWhitespace)
#c1$content[[1]][[1]]

c1 <- tm_map(c1, stemDocument, language = "english")
c1$content[[1]][[1]]

# WordCount über Corpus
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))

unigrams = DocumentTermMatrix(c1)
bigrams  = DocumentTermMatrix(c1, control = list(tokenize = BigramTokenizer))

# Ergebnisse sichten
dimnames(unigrams)$Terms
length(dimnames(unigrams)$Terms)

# Hier kann die Gesamtzahl der zu betrachtenden Wörter eingegrenzt werden (je nach
Auftretenshäufigkeit in allen Dokumenten)
s.unigram_nongerman = removeSparseTerms(unigrams, 0.98)
s.bigram_nongerman  = removeSparseTerms(bigrams, 0.98)

s.unigram_nongerman = data.matrix(s.unigram_nongerman)
s.bigram_nongerman  = data.matrix(s.bigram_nongerman)

m = as.data.frame(cbind(s.unigram_nongerman, s.bigram_nongerman))
colnames(m)


# ---------------TF-IDF Normierung:-------------

docsize = rowSums(m*(m > 0))
Y_TFIDF = m / docsize*log(nrow(m) / (1 + rep(colSums(m > 0), each = nrow(m)))) * (m > 0)

# nans entfernen
Y_TFIDF = na.omit(Y_TFIDF)


# Summe aus allen Wortzählungen für Word Cloud
termDocMatrix = t(as.data.frame(apply(Y_TFIDF, 2, sum)))

termDocMatrix = as.data.frame(termDocMatrix)
colnames(termDocMatrix) = colnames(m)

# Word Cloud erstellen
ap.m <- t(as.matrix(termDocMatrix))
```

```
ap.v <- sort(rowSums(ap.m), decreasing=TRUE)
ap.d <- data.frame(word=names(ap.v), freq=ap.v)
table(ap.d$freq)
pal2 <- brewer.pal(8, "Dark2")
png("d:/wordcloud norm.png", width=1600, height=1200)
wordcloud(ap.d$word, ap.d$freq, scale=c(12,.4), min.freq=5,
          max.words=300, random.order=FALSE, rot.per=.15, colors=pal2)
dev.off()
```

Führt man den obigen Code aus, so erhält man die Wordcloud, welche die häufigsten Wörter im gesamten Datenbestand animiert aufzeigt (Abb. 2).

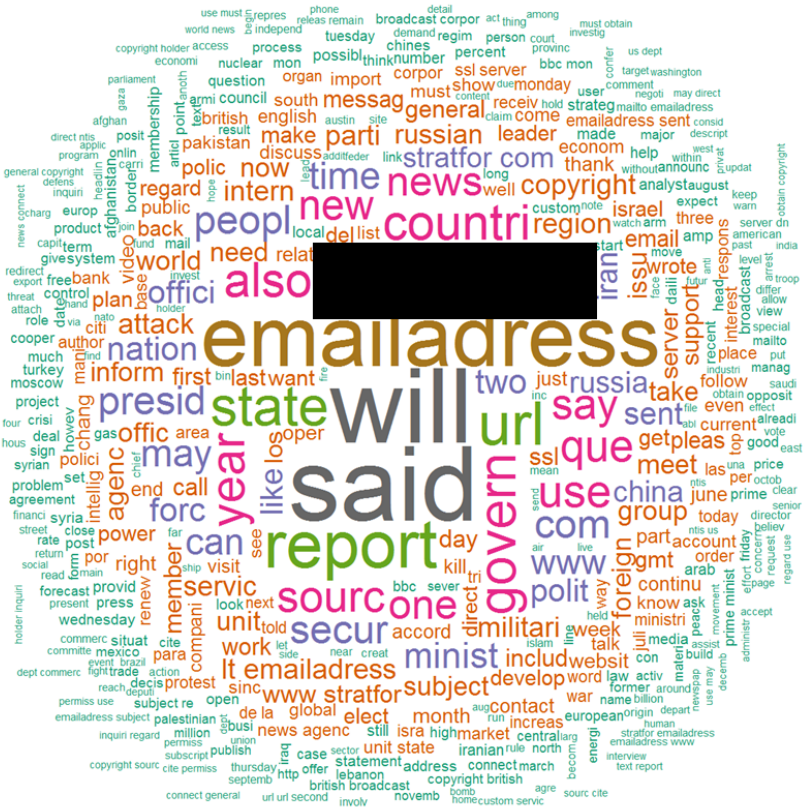


Abbildung 2: Word Cloud für den gesamten Dokumentenbestand. Der Firmenname wurde unkenntlich gemacht.[1]

Die Clusteranalyse

Unabhängig vom Grad der Übereinstimmung mit der verwendeten (und auch jederzeit anpassbaren) Selektoren-Liste, kann man die Texte nach thematischer Zusammengehörigkeit clustern, also ähnlichen Gruppen zuordnen. Dieses kann auf unterschiedliche Art und Weise erfolgen und bildet ein umfangreiches eigenes Themengebiet. Aus diesem Grund werden hier nur die wesentlichen Elemente des Vorgehens vorgestellt und als Code-Beispiel präsentiert.

[1] Für die Wordcloud wurde in R das Package ‚Wordcloud' verwendet, welches alle Rechte zur nicht-kommerziellen Weiterverbreitung einräumt.

Zu den performantesten und auch am häufigsten eingesetzten Clusteralgorithmen gehört der *k-means* Algorithmus. Dabei werden zufällige vorher definierte *k* „Zentren“ in den Daten ausgewählt und iterativ so lange verschoben, bis die Summe aller „within-cluster-distances“ minimiert ist und gleichzeitig der Abstand zwischen den Clustern untereinander maximiert. In unserem Fall hätten wir eine *m x n*-dimensionale Datenmatrix mit ebenso vielen Dimensionen wie gefundenen Wörtern und so vielen Beobachtungen wie Dokumenten (Texten).

Vorab muss jedoch beachtet werden, dass die *m zu n*-Beziehung niemals kleiner als 1:30 werden darf, will man aussagekräftige Ergebnisse bekommen. Die Anzahl an Dimensionen muss also drastisch reduziert werden, da in unserem Fall 1500 Wörter (*m*) rund 12.000 Texten (*n*) gegenüberstehen - eine Beziehung von nur 1:8.

Eine Dimensionsreduzierung erfolgt mithilfe der Hauptkomponentenanalyse, auf welche an dieser Stelle in der Theorie nicht näher eingegangen wird. Wichtig ist, dass im Ergebnis die verwendeten Daten so „umgestellt“ werden, dass mit rund 200 Dimensionen, also den ersten 200 Hauptkomponenten, rund 50 % der Varianz erklärt wird, was für das gezeigte Beispiel vollkommen ausreicht, wir erreichen also eine *m zu n* Beziehung von 1:60 und können die Daten folglich einer Clusterung unterziehen.

```
# Hauptkomponentenanalyse (PCA)

# Standardisieren
Y_TFIDFstand <- scale(Y_TFIDF)
# checken, ob Mittelwert = 0 und Stand.abw. = 1
colMeans(Y_TFIDFstand)
apply(Y_TFIDFstand, 2, sd)

# stats package
#sichergehen, ob kein NA vorhanden
Y_TFIDFstand[which(is.na(Y_TFIDFstand))]

# PCA berechnen
pca1 = prcomp(Y_TFIDFstand, scale = FALSE)

# loadings (Coefficients)
head(pca1$rotation)

# scores (PCs)
head(pca1$x)
pca1$x[1:10, 1:3]

# variance
variance = pca1$sdev^2 / sum(pca1$sdev^2)

for (i in 1:length(pca1$x[ ,2]))
{
  b = sum(variance[1:i])

  if (b*100 > 40) # hier die zu erklärende Varianz in % rein
  {
    print(i)
    break
  }
}

scores = pca1$x
```

Im Anschluss kann mit dem Ergebnis aus *i* des obigen Codeabschnitts die Clusteranalyse auf den sogenannten *„Scores“* durchgeführt werden. Die Scores sind die jeweiligen Hauptkomponenten aus der PCA und zwar absteigend in Reihenfolge der erklärten Varianz. Hier werden lediglich die ersten 1 bis *i* Spalten (Dimensionen) herangezogen, da diese für die entsprechende Varianzerklärung ausreichen.

Da die Anzahl *k* im Vorfeld festgelegt werden muss, man jedoch noch keine Info über die passendste Anzahl an Clustern hat, können iterativ für jede Anzahl *k* zwischen 2 und 100 ausgewählte Indizes zur Güte errechnet werden, auf diesen Teil wird jedoch an dieser Stelle verzichtet und stattdessen der Einfachheit halber mit einer festgelegten Menge k=80 gearbeitet.

```
# K Means mit bestmöglicher Clusteranzahl für ganzen Datensatz berechnen (hier 60)
k=80
result KMeans = kmeans(scores[ ,1:200], k, iter.max = 100, nstart = 10)

g = g[c(row.names(scores)), ]
g$Cluster = as.factor(result KMeans$cluster)

# table of sample
table(result KMeans$cluster)
table(result KMeans$cluster) / length(scores[ ,1])*100
idx1 = g$Cluster

# Daten rauschreiben
write.csv(g, 'D:/Indizes Cluster.csv')



# ----------------------------------
# Alternativ: 'cosine' distance anstatt 'euclidean'
library(akmeans)
k=60
result KMeans cos  =  norm.sim.ksc(scores[,  1:200],  k,  init.cen  =  NULL,  init.mem  =  NULL,
iter.max = 100)
idx2 = result KMeans cos$cluster

# table of sample
table(result KMeans cos$cluster)
table(result KMeans cos$cluster) / length(idx2)*100



#----------------------------------------
# wordcloud pro cluster
library(wordcloud)

for (cluster in 1 : k)
{
  termDocMatrix = t( as.data.frame(apply(Y TFIDF[which(idx2==cluster), ], 2, sum)) )

  termDocMatrix = as.data.frame(termDocMatrix)
  colnames(termDocMatrix) = colnames(m)

  # Word Cloud erstellen
  ap.m <- t(as.matrix(termDocMatrix))
  ap.v <- sort(rowSums(ap.m), decreasing=TRUE)
  ap.d <- data.frame(word=names(ap.v), freq=ap.v)
  table(ap.d$freq)
  pal2 <- brewer.pal(8, "Dark2")
  png(paste0("d:/wordcloud ", cluster, ".png"),  width=800, height=600)
  wordcloud(ap.d$word, ap.d$freq, scale=c(12,.4), min.freq=3,
            max.words=150, random.order=FALSE, rot.per=.15, colors=pal2)
  dev.off()

  print(cluster)
}
```

Jedem Text (also jeder Zeile im data.frame *g*) ist nun genau einem Cluster zwischen 1 und 80 zugeordnet. Möchte man nun wissen, welche Eigenschaften und Inhalte die Cluster ausmachen oder inwieweit Cluster sich ähneln, so berechnet man zunächst die Wordcloud für jedes Cluster. In Abb. 3 sind exemplarisch 3 der 80 Cluster dargestellt, man sieht, dass sich gewisse „Themen“ herauskristallisieren, es also offensichtlich in den Texten, welche dem Cluster 31 zugeordnet sind, um Energie, Öl und Ähnliches geht.

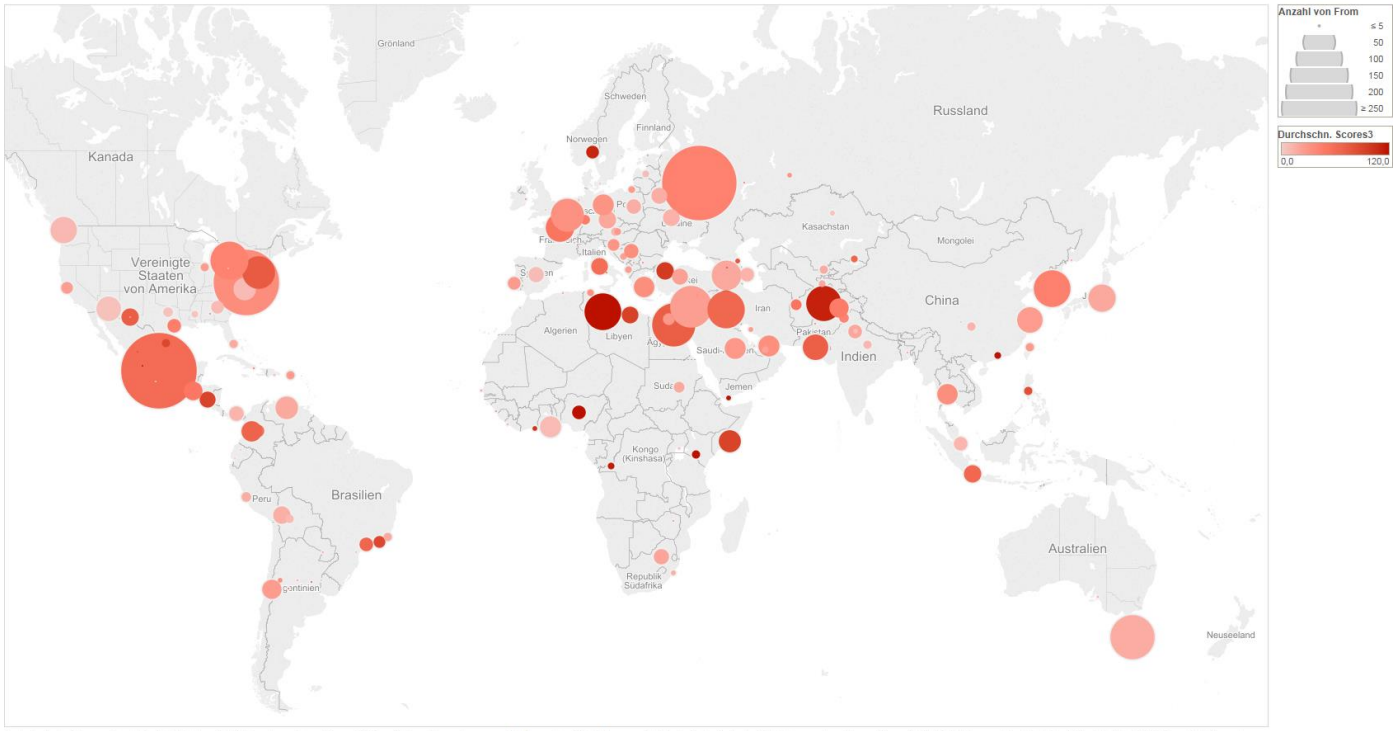
Abbildung 3: Exemplarische Auswahl von drei Clustern in Wordcloud-Darstellung.[2]

Visualisierung der Ergebnisse und Interpretation

Zu den wichtigsten Ergebnissen bei der Auswertung von Kommunikationskanälen gehört die Reduzierung auf die wesentlichsten Merkmale des Datensatzes, welcher in der Realität gern in Sphären jenseits der 100 Mio. Datensätze gehen kann. Was getan wird, ist in erster Linie das Herunterbrechen des gesamten Datensatzes auf ein paar hundert bis tausend „verdächtige“ Elemente, die am Ende fast immer „von Hand“ analysiert werden müssen. Der Algorithmus und die Auswertungsroutine macht also im Prinzip nichts anderes, als die Wahrscheinlichkeit zu erhöhen, die verdächtigen Elemente aufzufinden und als solche vorab zu kennzeichnen.

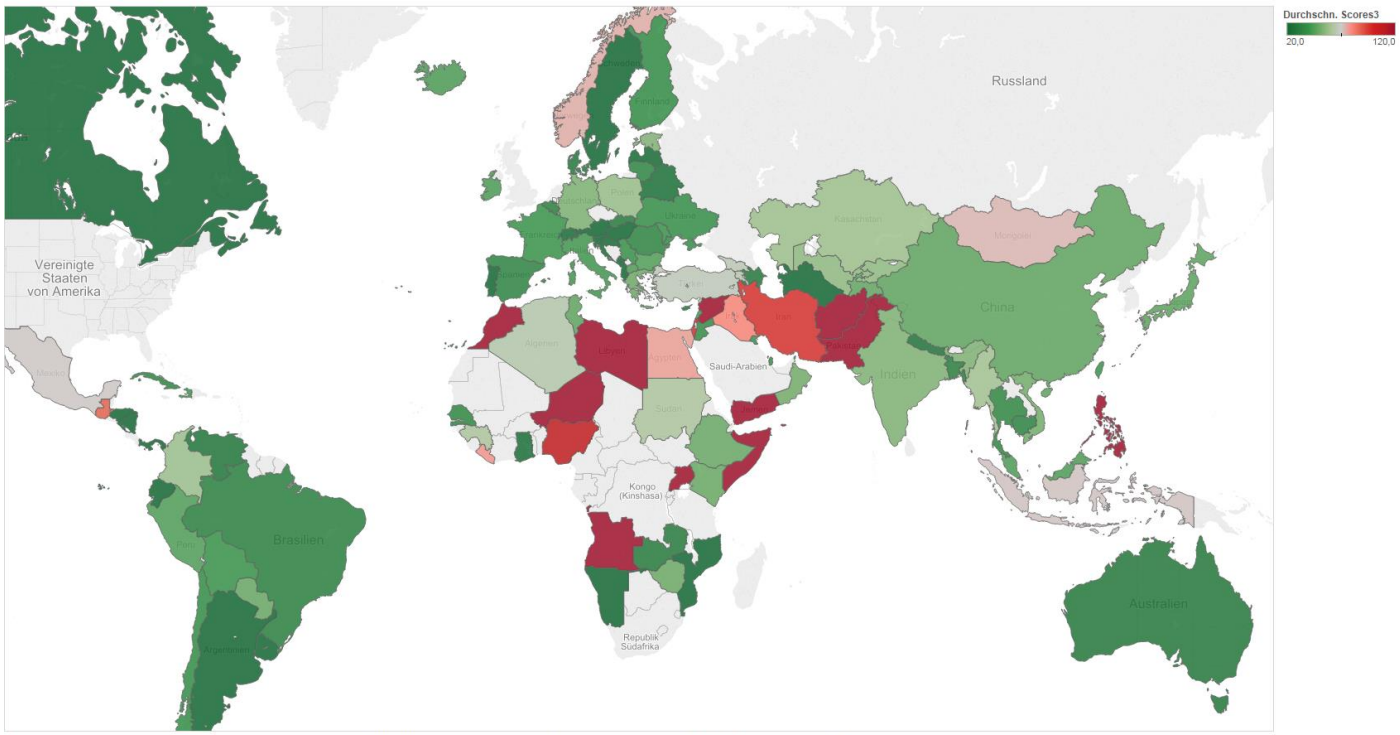
Die Analyse lässt sich anschließend am besten durch übersichtlich gestaltete Grafiken und/ oder Dashboards durchführen. Wie man solche gestalten kann (hier wurde Tableau Software verwendet), wird im Folgenden näher aufgezeigt.

[2] Für die Wordcloud wurde in R das Package ‚Wordcloud‘ verwendet, welches alle Rechte zur nicht-kommerziellen Weiterverbreitung einräumt.



Karte basiert auf Laenge1 und Breite1. Farbe zeigt Mittelwert von Scores3 an. Größe zeigt Anzahl von From an. Details werden für City1 angezeigt. Die Ansicht wird unter City1, Laenge1 und Ausschlüsse (Breite1,City1,Laenge1) gefiltert. Der Filter City1 behält 275 von 672 Elementen bei. Der Filter Laenge1 behält nur Nicht-NULL-Werte bei. Der Filter Ausschlüsse (Breite1,City1,Laenge1) behält 671 Elemente bei.

Abbildung 4: Weltkarte mit Kennzeichnung der Städte, die im jeweiligen Text am häufigsten vorkommen. Die Größe der Blasen spiegelt die Anzahl an Nachrichten pro Stadt wider, die Farbe den Mittelwert aus Score3 (s.o.), also die Übereinstimmung mit den Selektoren. Dunkelrot markiert sind Nachrichten, mit hoher Trefferquote zu terrorrelevanten Begriffen.



Karte basiert auf Längengrad (generiert) und Breitengrad (generiert). Farbe zeigt Mittelwert von Scores3 an. Details werden für Countries angezeigt. Die Ansicht wird unter Countries gefiltert, wobei 114 von 156 Elementen beibehalten wird.

Abbildung 5: Weltkarte mit Kennzeichnung der Länder, die im jeweiligen Text am häufigsten vorkommen. Die Farbe spiegelt den Mittelwert aus Score3 (s.o.) wider, also die Übereinstimmung mit den Selektoren. Dunkelrot markiert sind Nachrichten, mit hoher Trefferquote zu terrorrelevanten Begriffen.

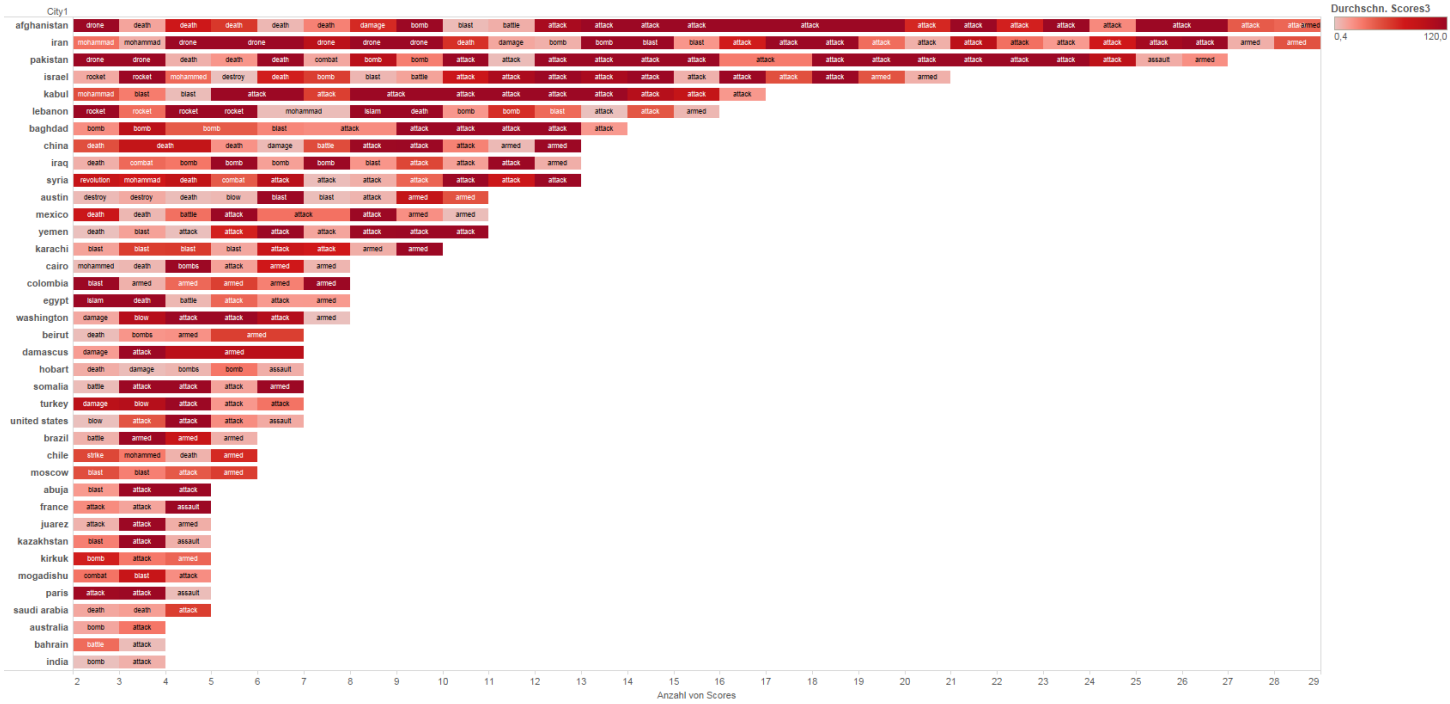


Abbildung 6: Genannte Städte weltweit, die im Zusammenhang mit Nachrichten auftauchen, die insgesamt den höchsten Score3-Wert (Übereinstimmung mit Selektorenliste) aufweisen. Jeder Unterteilung der Balken steht für eine E-Mail-Adresse, welche hier ausgeblendet ist, und einem jeweils am häufigsten auftauchenden Begriff dazu.

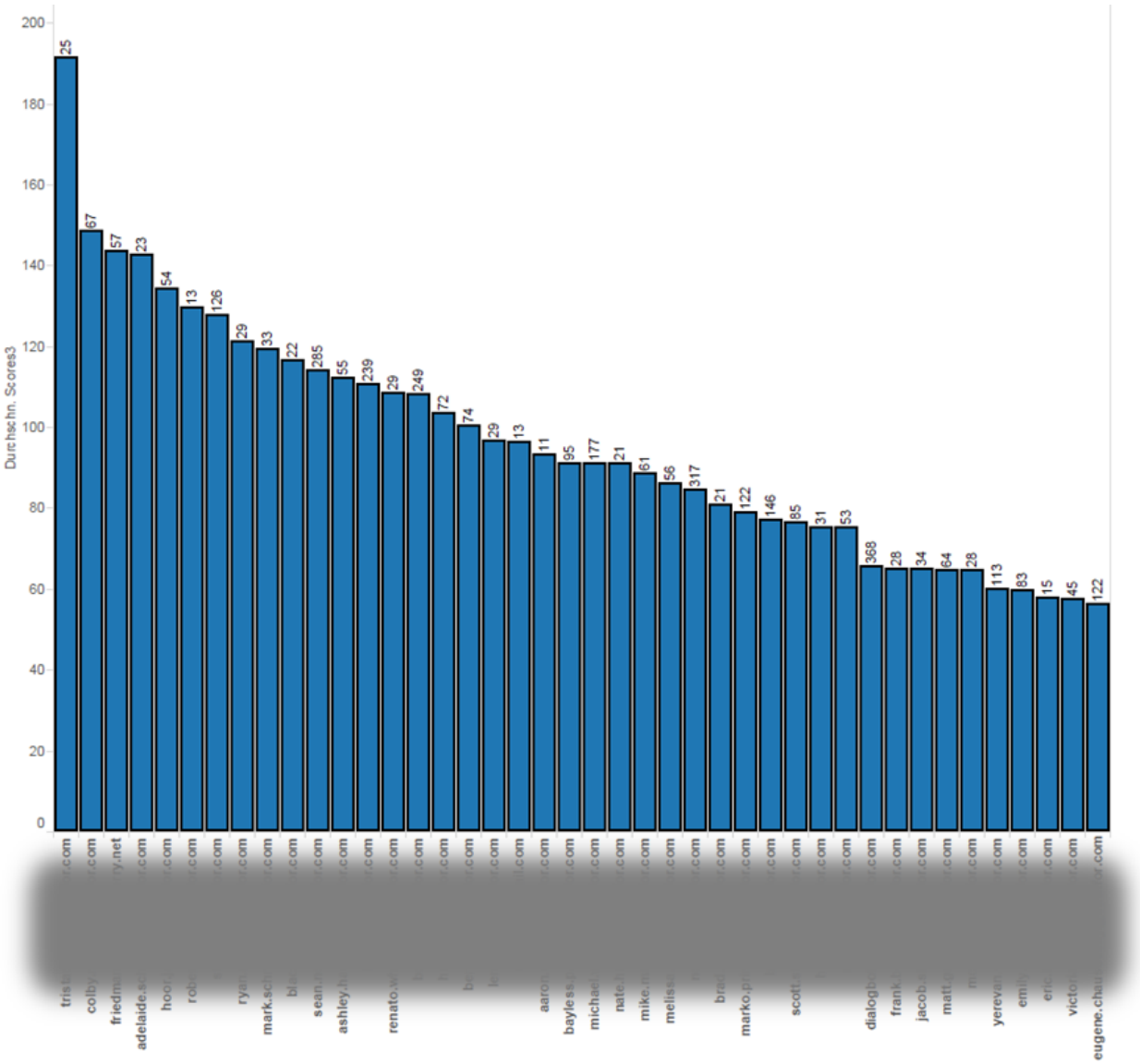


Abbildung 7: Top Email-Adressen mit dem höchsten Score3-Wert aus allen verschickten Nachrichten. Zusätzlich ist die Anzahl an verschickten Nachrichten über den Balken abgebildet.

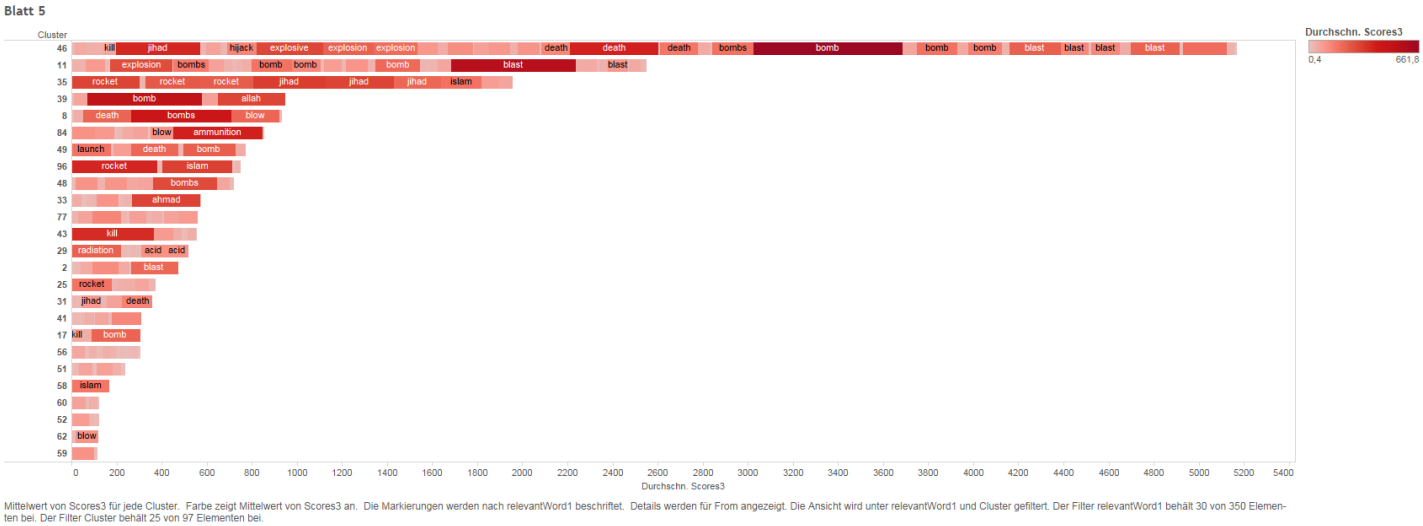


Abbildung 8: Cluster mit den höchsten mittleren Score3-Werten, unterteilt in die jeweils am häufigsten auftauchenden Wörter pro eMail-Adresse (ausgeblendet).

Zusammenfassung und Fazit

Die analytische Auswertung von Texten in großen Dokumentenbeständen gehört zum Tagesgeschäft von Geheimdiensten und anderen Überwachungs-Institutionen. Wie dabei die Praxis aussehen könnte, wurde im gezeigten Beispiel anhand frei zugänglicher Kommunikationsdaten (E-Mail Verkehr innerhalb einer großen Organisation) aufgezeigt, ohne jedoch den Anspruch zu erheben, die tatsächliche Praxis und Fachlichkeit im Detail zu kennen.

Des Weiteren wurde aufgezeigt, wie man mit einfachen Mitteln und ein wenig Programmiererfahrung in der Lage ist, eigene Analysen mit der Open Source Sprache *R* zu starten und mithilfe moderner Visualisierungstools (hier Tableau Software) grafisch so aufzubereiten, dass die relevantesten Informationen auf einem Blick ersichtlich werden, indem man z.B. sie auf die kleinsten Einheiten (*wer ist verdächtig?)* herunterbricht.

Im Ergebnis gelangt man ausgehend von 50.000 Texten zu einigen wenigen „verdächtigen“ Texten, die bestimmte Übereinstimmungen mit jederzeit veränderbaren „Selektoren“ aufweisen. Über Clusterbildung werden zusätzlich ähnliche Texte nach Themengebieten zusammengefasst, so dass nun themenbezogene Auswertungen möglich sind. Verknüpft man diese Daten mit einer Historie, die man bspw. bereits zu bestimmten Usernamen, Klarnamen oder bestimmte Browser User Agents gesammelt hat, so ergeben sich immer klarere Bilder von User(n) (-Gruppen). Über diese Charakterisierung lassen sich wiederum neue Muster in den Daten suchen, die jeweils Ähnlichkeiten zu den erkannten Mustern aufweisen, was viele neue Optionen der potenziellen Gefahren-Identifizierung mit sich bringt.