Sinn verstehen statt Wörter analysieren

Text-Mining

Lisa Wagner

Text-Mining-Algorithmen erfassen Textinhalte, gruppieren große Mengen an Dokumenten automatisch nach Themen, finden verwandte Texte und bereiten Kernbegriffe visuell auf. Jedes Projekt ist ein Unikat: Datenanalysen fokussieren auf relevante Informationen für ihre Fragestellungen. Dazu braucht es die richtigen Anpassungen: Der Artikel zeigt die konkreten Schritte.

Ob Kundenrezensionen, Erfahrungsberichte, Tweets oder Facebook-Einträge – Unternehmen sind mit einer schier unerschöpflichen Menge an Textinformationen konfrontiert. In ihren internen Systemen häufen sich Support-Tickets, Testfallbeschreibungen und Log-Einträge. Niemand kann diese Menge an Texten noch sichten, um die wichtigsten Punkte und interessante Auffälligkeiten zu extrahieren.

Eine ganze Reihe von automatisierten und semi-automatisierten Big-Data- und Data-Mining-Verfahren bringen Ordnung und Übersicht in die großen Datenmengen. Sie basieren auf Algorithmen, die sich nur auf Zahlen verstehen. Das heißt, sie stellen jeden Datenpunkt als (hochdimensionalen) Vektor aus Zahlen dar und kalkulieren dann zum Beispiel Abstände oder Winkel zwischen ihnen. Aber welche Zahlen nimmt man für Dokumente?

Text-Mining-Algorithmen basieren auf dem Vorkommen von bestimmten Wörtern. Einzelne Wörter geben einen guten Eindruck über das behandelte Thema: Ein Text, der 30-mal das Wort „Kaffeemaschine“ und einmal das Wort „Elefant“ enthält, handelt wahrscheinlich von einer Kaffeemaschine. Die Algorithmen werten Dokumente aus, die Summe der Dokumente nennt man Text-Corpus. Die für das Mining-Projekt verwendeten Wörter werden als Features bezeichnet; die Ausprägung der Features beschreibt einen Text als Datenpunkt.

Also einfach zählen, wie oft jedes Wort vorkommt. Kein Problem, oder? Leider doch: Die deutsche Sprache hat einen Standardwortschatz von circa 75.000 Wörtern. Ein Vektor mit so vielen Dimensionen würde zu unvertretbar langen Laufzeiten bei der Berechnung und Verwendung der Algorithmen führen. Außerdem benötigt man eine unglaublich große Menge verschiedener Dokumente, damit sinnvolle Modelle herauskommen. Als Faustregel zielt man auf maximal 300 Dimensionen. Die Kunst ist, geeignete Wörter festzulegen und richtig zu zählen.

Der Artikel zeigt das Vorgehen an einem Praxisbeispiel: Ein Kaffeemaschinen-Hersteller möchte das Kunden-Feedback aus Online-Produktreviews, Support-Anfragen, Schadensmeldungen und anderen Quellen auswerten. Die Herausforderung dabei: Einige Texte sind in der Alltagssprache geschrieben, mit umgangssprachlichen Begriffen und inkonsistenter Wortwahl. Für ein sinnvolles Ergebnis müssen die Daten zunächst aufbereitet werden: Man bereinigt den Datensatz von sprachlichen Ungenauigkeiten und überflüssigen Informationen. Erst die bereinigten Daten werden ausgewertet. Für reine Mining-Projekte verwendet man dabei in der Regel R oder Python. Im Java-Umfeld bietet sich Spark an. Zunächst beschreibt der Artikel die Datenbereinigung, anschließend geht es um das eigentliche Data-Mining.

Brühen, brühte, gebrüht: Stemming bündelt nach Wortstamm

Füllwörter gibt es in jedem Text: Artikel, Pronomen und Präpositionen gehören zum Beispiel dazu, Konjunktionen und Hilfsverben ebenso. Diese „kleinen“ Wörter, im englischen „stop words“ genannt, transportieren kaum für die Analyse relevanten Inhalt, machen aber einen beträchtlichen Teil von Texten aus. Damit sie die Auswertung nicht dominieren, werden sie gleich zu Beginn herausgefiltert. Stop-Word-Lexika, mit denen man diese Wörter gezielt herausfiltern kann, gibt es für die meisten Sprachen.

Anschließend bereitet man den Text so auf, dass der Algorithmus Worthäufigkeiten erkennen kann: Man trägt in diesem Schritt der Tatsache Rechnung, dass Grundwörter an Zeit, Genus oder Person angepasst werden. Im Deutschen ist das besonders stark ausgeprägt: Damit die gebeugten Formen von „brühen“ in den Sätzen „Die Kaffeemaschine brüht lange“, „Das Brühen dauert lange“ und „Die Maschine hat zu lange gebrüht“ nicht als einzelne Wörter gezählt werden, kommt das sogenannte Stemming zum Einsatz: Dabei werden nach festen, von der jeweiligen Sprache abhängigen Regeln alle gebeugten Präfixe und Suffixe entfernt, bis nur noch der Wortstamm übrig bleibt – in unserem Fall „brüh“.

Es gibt auf die Sprache abgestimmte Algorithmen, sie arbeiten jedoch bei einer komplizierten Sprache wie Deutsch nicht fehlerfrei. Besonders Lautveränderungen bei starken Verben werden nicht immer richtig zugeordnet. Die Sätze „Die Kaffeemaschine läuft beim Entkalken über“ und „Die Kaffeemaschine lief beim Entkalken über“ schildern das gleiche Problem, der gemeinsame Wortstamm der Verbformen von „laufen“ wird aber nicht erkannt. Deshalb werden wichtige Wörter über eine Synonymliste auf ihren Stamm abgebildet.

Ist Geschmack Aroma oder Stil? Individuelle Synonymlisten erstellen

„Die Satzschublade klemmt oft“ und „Das Satzfach ist schwergängig.“ – Synonyme erschweren das automatisierte Erfassen davon, wie häufig ein Sachverhalt im Text beschrieben oder angesprochen wird. Für die Auswertung bildet man Synonyme aufeinander ab und ergänzt Verbformen von starken Verben. Außerdem kann man Dialekt-Begriffe aufeinander abbilden, zum Beispiel das „Haferl“ auf die „Tasse“.

Vorgefertigte Synonymlexika sind nicht zu empfehlen, da sie Wörter austauschen, ohne den Kontext zu würdigen. Im Satz „Die Maschine hat Stil“ könnte ein Synonym-Wörterbuch zum Beispiel das Wort „Stil“ mit Geschmack ersetzen. In Reviews von Kaffeemaschinen wird sich das Wort „Geschmack“ jedoch häufig auf das Kaffeearoma beziehen. Die unterschiedlichen Bedeutungen gehen so verloren, es entstehen Sinnverschiebungen. Stattdessen wird die Synonymliste für jedes Text-Mining-Projekt individuell gepflegt.

Zusätzlich bildet man Wörter ab, die nur in einem bestimmten Kontext synonym verwendet werden. Es kann etwa sein, dass die Kaffeemaschine über eine rote LED verfügt, die bei Fehlermeldungen leuchtet. Daher kann bei Reviews davon ausgegangen werden, dass „Warnlämpchen“ und „rote LED“ das gleiche bezeichnet, auch wenn die beiden Begriffe in anderen Zusammenhängen keine Synonyme sind.

Die manuelle Zuordnung verhindert verfälschte Ergebnisse: Das Aussehen der Kaffeemaschine bekommt so keinen Geschmack; es wird klar, in wie vielen Fällen Kunden über die klemmende Satzschublade schreiben; und der Themencluster „Warnlämpchen“ enthält auch die Nennungen roter LEDs.

Schwarz und Weiß: Blacklists und Whitelists pflegen

Kleine, sinnleere Wörter werden über Stop-Word-Lexika aus der Analyse zwar ausgeschlossen, doch andere Wörter bleiben unberücksichtigt. Blacklists schließen themenbedingt bestimmte Wörter aus der Analyse aus. Auch sie werden für jedes Text-Mining-Projekt individuell gepflegt. So können zum Beispiel Grußfloskeln aus der Analyse ausgenommen werden, wenn Kunden-E-Mails zum Text-Corpus gehören. Im Text-Mining-Projekt eines Kaffeemaschinen-Herstellers lohnt es sich zudem, das Wort Kaffeemaschine auszuschließen, da es wahrscheinlich in fast jedem Produktreview, in fast jeder E-Mail und in nahezu allen Support-Notizen vorkommt.

Statt einer Blacklist kann eine Whitelist vorgeben werden: Die Liste mit explizit interessanten Wörtern macht Cluster-Algorithmen stabiler und aussagekräftiger. Eventuell lohnt sich die Kombination mit einer gut gepflegten Synonyme-Liste. Möchte zum Beispiel der Kaffeemaschinen-Hersteller herausfinden, wie das Design seines neuen Modells beim Kunden ankommt, legen die Daten-Analysten eine Whitelist mit Begriffen an, die Form, Farbe und Bedienbarkeit beschreiben.

Voraussetzung für eine aussagekräftige Antwort ist eine sorgfältige und unvoreingenommene Wortauswahl. Beim Arbeiten mit einer Whitelist bedeutet das: Enthält die Liste alle relevanten Begriffe aus allen relevanten Untergruppen? In unserem Beispiel geht es neben Form und Farbe auch um die Bauart (Ist der Ein- und Ausschalter gut erreichbar?) und die Bedienbarkeit (Verstehen Kunden die Anweisungen auf dem Display?). Wurden wichtige Schlagwörter vergessen, verliert die Auswertung an Aussagekraft. Zudem werden Daten-Analysten durch ihre Erwartungen an den Inhalt der Dokumente und den Fokus der Analyse bei der Wahl der Schlagwörter beeinflusst.

Eine erste Datengrundlage für die Whitelist kann eine Auswertung aller im Text-Corpus verwendeten Wörter sein. Sie zu erstellen, verursacht wenig Aufwand; das Aussortieren und Gruppieren von Begriffen für die Synonymliste schon mehr. Die Arbeit lohnt sich, wenn die Antwort auf eine konkrete Frage gesucht wird, zum Beispiel: „Wie gefällt Kunden das neue Design der Maschine?“. Unerwartete Phänomene, etwa Probleme mit dem Entkalkungsmechanismus, deckt man so nicht auf. Dafür ist eine Analyse ohne Whitelist besser geeignet.

Geeignete Begriffe für beide Listen, Blacklist wie Whitelist, sind nie offensichtlich. Sie entstehen in der Regel in mehreren Arbeitsschritten: Man startet mit einer initialen Liste – die sogar leer sein kann – und analysiert den Text-Corpus in mehreren Iterationen. So schärft man das Ergebnis, bis man ein sinnvolles Resultat hat. „Sinnvoll“ ist dabei ein relatives Maß: Ist die Fragestellung beantwortet? Lassen sich Cluster thematisch voneinander abgrenzen? Eine Daumenregel gibt es nicht.

```
// bereitet den Text-Corpus vor, indem alle unerwünschten Wörter
// entfernt und die anderen vereinheitlicht werden
prepareCorpus ( textCorpus, language, optional customSynonymList,
  optional customBlackList, optional customWhiteList )
for( text in textCorpus )
    // kann für alle Texte parallel ausgeführt werden
  text.delete( stopWordDict( language ) )
  text <- stemmingAlgo( text, language )
  if( exists( customSynonymList ) )
    text.replace( customSynonymList )
  if( exists( customBlackList ) )
    text.delete( customBlackList )
  if( exists( customWhiteList ) )
    // es werden alle Wörter entfernt, Whitelist ausgenommen
    text.delete( ! customWhiteList )
// die Texte bestehen jetzt nur noch aus auswertungsrelevanten Wörtern
return textCorpus
```

Pseudo-Code 1: Bereinigung des Text-Corpus

Häufigkeitsgrenzen steuern die Anzahl der Schlagwörter

In den vorangegangenen Arbeitsschritten haben wir sinnleere Wörter aus der Auswertung ausgenommen, Wörter nach Wortstämmen gruppiert, Synonyme aufeinander abgebildet und im Projektkontext irrelevante Wörter mit Blacklists ausgeschlossen. Üblicherweise verbleiben nach diesen Schritten immer noch weit mehr Wörter als Analyse-Dimensionen gewünscht sind. Zur Erinnerung: Mehr als 300 Features sind nicht ratsam.

Allerdings sind auch nicht alle Wörter hilfreich: Ein Wort, das in fast jedem Dokument des Text-Corpus vorkommt (mehr als 95 Prozent), macht Dokumente nicht unterscheidbar. Wörter, die ganz selten und in sehr wenigen Dokumenten vorkommen (weniger als 5 Prozent), verschleiern dagegen schnell Ähnlichkeiten zwischen Dokumenten.

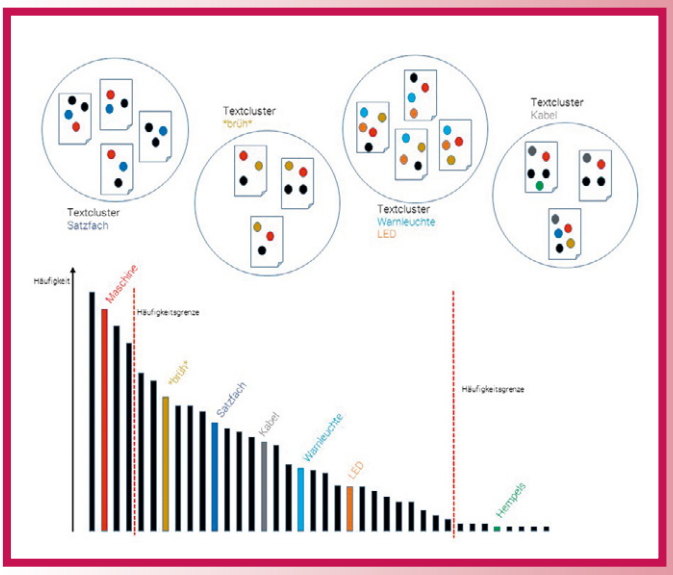

Abb. 1: Häufigkeitsanalyse: Welche Worte sind für das Clustering hilfreich?

Zwei Rezensionen mit ähnlichem Feedback werden potenziell als unterschiedlich angesehen, wenn eine von beiden den Kommentar enthält „Die Kabel sind unschön verlegt. Es sieht aus wie bei Hempels unterm Sofa“. Dieses Dokument enthält mit „Hempels“ und „Sofa“ Wörter, die vermutlich in keinem anderen Dokument über Kaffeemaschinen vorkommen. Dabei verändern beide Wörter die Aussage des Dokuments nicht. Der Fall an sich – die unschöne Kabelführung – würde trotzdem über das Feature „Kabel“ entdeckt. Auch bei einer Analyse mit Hilfe von Whitelists empfiehlt es sich, sehr häufige und sehr seltene Wörter zusätzlich auszuschließen.

Zu häufige oder zu seltene Wörter sollten deshalb für die allgemeine Analyse aussortiert werden (s. Abb. 1). Die nun noch verbleibenden Wörter bilden das Feature-Set für das eigentliche Data-Mining.

```
// bestimmt die Features anhand der absoluten oder relativen
// Häufigkeiten der Wörter oder betroffenen Dokumente
computeFeatures( textCorpus, min, max, absNumbers, asWordCount)
// finde alle potenziellen Features
allTexts <- ""
for( text in textCorpus )
  allTexts.add(text)
featureVector <- allTexts.split(" ")
featureVector.deleteDuplicates
// zähle Häufigkeiten
countsVector <- (featureVector, 0)
for( text in textCorpus )
    //kann parallelisiert werden, wenn counts am Ende addiert werden
  for( word in featureVector )
    anz <- text.count(word)
    if( asWordCount )
      countsVector(word) <- countsVector(word)+anz
    else if ( anz > 0 )
      countsVector(word)++
// vergleichen mit min und max
for( word in featureVector )
  if( absNumbers )
    if( countsVector(word) < min or countsVector(word) > max )
      featureVector.delete(word)
    else
      if( asWordCount )
        total <- sum ( for(text in textCorpus) text.length )
      else
        total <- textCorpus.size
      if( countsVector(word) / total <
          min or countsVector(word) /total > max )
        featureVector.delete(word)
return featureVector
```

Pseudo-Code 2: Bildung des Feature-Sets

Einfach nur zählen?

Das Feature-Set ist festgelegt: Jetzt muss für jedes Dokument im Text-Corpus die Ausprägung der Features bestimmt werden, um sie als Datenpunkte zu beschreiben. Alle verwendeten Algorithmen arbeiten mit Worthäufigkeiten, die für die Analyse als Features festgelegt wurden. Verschiedene Arten zu zählen beeinflussen das Ergebnis maßgeblich:

- *Absolute Anzahl:* Sie sagt aus, wie oft jedes Feature in dem Dokument absolut vorkommt. Das Vorgehen steht für die einfachste und offensichtlichste Methode. Jedoch eignet es sich nur für sehr lange Texte mit vergleichbarem Umfang. Für kurze Texte sind absolute Werte nicht zu empfehlen, da der Unterschied zwischen „kommt gar nicht vor" und „kommt einmal vor" genauso stark gewichtet wird wie einmaliges Vorkommen im Vergleich zum zweimaligem.
- *Abnehmender Einfluss:* Oft ist es entscheidend, ob ein Wort einmal oder mehrmals in einem Dokument vorkommt. Mindestens genauso wichtig ist der Unterschied zwischen Features mit einer Nennung und gar keiner. Der Unterschied zwischen 73 und 74 Nennungen ist dagegen in den meisten Fällen vernachlässigbar. Dieses Phänomen fängt man mit einer Funktion über die absolute Anzahl der Nennungen ab, die für kleine Werte stark wächst und dann immer stärker abflacht – zum Beispiel eine Wurzel hoher Potenz. Sie verdeutlicht die Relevanz des Wortes in der Feature-Ausprägung. Die Methode eignet sich am besten, wenn die Häufigkeit der vorkommenden Wörter mit einbezogen werden soll. Die Texte sollten in etwa gleich lang sein. Über die Wahl der Funktion kann gesteuert werden, wie stark der zusätzliche Einfluss von hohen Werten abnehmen soll.
- *Vorkommen oder nicht:* Die Unterscheidung zwischen Dokumenten mit und ohne Schlagwort-Nennung ist der Extremfall von abnehmendem Einfluss. Es wird nur unterschieden, ob die Features in den Dokumenten enthalten sind oder nicht. Diese Vorgehen empfiehlt sich, wenn die Worthäufigkeit >0 gar keinen Einfluss haben soll. Ist das Feature-Set gut gewählt, können die Dokumente unterschiedlich lang sein.
- *Wortverhältnisse:* Hier untersucht man, in welchem Verhältnis Wortanzahl und Textmenge stehen. Das gibt einen Hinweis auf die Relevanz eines Wortes in einem Text. Eine zweizeilige Support-Notiz, in der das Wort „Mahlwerk" zweimal auftaucht, beschreibt vermutlich ein Problem beim Kaffeemahlen. Bei zweimaliger Nennung in einem einseitigen Text kann es auch um ein anderes Thema gehen. Man vergleicht hier entweder gegen die Gesamttextlänge (Normierung) oder nur gegen die anderen Schlagworte (Cosinus-Ähnlichkeit). Diese Methode eignet sich besonders, wenn einerseits die Häufigkeit eines Schlagwortes von Wichtigkeit ist und andererseits Texte von sehr unterschiedlicher Länge verglichen werden. Sie ist jedoch rechenintensiver als andere Zählweisen.

Aus den Feature-Ausprägungen kann jetzt für jedes zu analysierende Dokument ein Vektor erstellt werden, der die Häufigkeiten der Features beschreibt. Dokumente, die Null-Vektoren erzeugen, also keines der Features enthalten, sollten als Outlier (Ausreißer) betrachtet und bei Bedarf gesondert behandelt werden. Sonst läuft man Gefahr, dass alle Texte, die vom Inhalt des Feature-Sets abweichen, als inhaltlich „gleich" betrachtet werden.

Thematisch gruppieren, Ähnlichkeiten finden, Neues einsortieren

Alle anderen Vektoren können mit verschiedenen Data-Mining-Verfahren analysiert werden: Clustering gruppiert sie thematisch; „Nächste-Nachbarn"-Algorithmen finden ähnliche Dokumente; und neue Dateien werden mit Hilfe von Klassifikatoren sinnvoll in vordefinierte Themenbereiche oder in gefundene Cluster einsortiert.

Um einen Text-Corpus in thematische Gruppen zu teilen, werden Clustering-Algorithmen verwendet. Dabei gibt es eine Reihe sehr unterschiedlicher Berechnungsmethoden. Ein paar Beispiele:

- support vector machines,
- K-means,
- neural networks,
- decision trees,
- Bayesian Clustering
- DBSCAN.

Ergebnis eines *Clustering-Algorithmus* ist die Zuordnung jedes Textes zu einem automatisch generierten Cluster. Manche Algorithmen ordnen Datenpunkte nicht zwangsweise einem Cluster zu; stattdessen können Datenpunkte als Outlier deklariert werden, die zu wenig Überschneidung mit anderen Daten haben. Andere Algorithmen geben statt der reinen Zugehörigkeit eine Wahrscheinlichkeitsverteilung über die Cluster an, zum Beispiel Bayesian Clustering. So kann man gut erkennen, ob ein Text genau in einen Themencluster passt oder zwischen zwei oder mehr Clustern liegt.

Die Suche nach nächsten Nachbarn zielt nicht darauf, Ordnung in den gesamten Text-Corpus zu bringen. Vielmehr findet man so ähnliche Texte zu einem Dokument. Das ist zum Beispiel bei der Themen-Recherche oder bei der Bearbeitung von Supporttickets von großer Hilfe. *Nächste-Nachbarn-Algorithmen* berechnen daher oft nur die Abstände zwischen dem Text und den Dokumenten im Corpus und geben entsprechend diejenigen mit dem geringsten Abstand aus. Diese Al-

Abb. 2: Wordclouds mit Maschine

Abb. 3: Wordclouds ohne Maschine

Diese Form der Darstellung gibt schnell einen Überblick über den Inhalt oder die am häufigsten behandelten Themen in den untersuchten Texten. Die Visualisierung macht die Auswertungen für Kollegen im Team oder aus der Fachabteilung leichter zugänglich. Mit Schlagworten lassen sich schnell erste Vergleiche zwischen verschiedenen Gruppen an Dokumenten ziehen.

gorithmen unterscheiden sich hauptsächlich darin, welches Distanzmaß (zum Beispiel die euklidische oder die Manhattan-Distanz) benutzt wird. Alternativ kann der Corpus normal geclustert werden; der gegebene Text wird einem der Cluster zugeordnet; als Nachbarn werden Vertreter aus dem Cluster angegeben. Das sind nicht zwangsläufig die nächsten Datenpunkte; oft ist etwas Variation in den Dokumenten hilfreicher als ein fast identischer Text, solange man sich im gleichen Themengebiet bewegt.

Dafür benutzt man einen weiteren Typ von Algorithmus: *Klassifikatoren* sortieren neue Datenpunkte in eine Gruppierung des Text-Corpus ein. Sie funktionieren ähnlich wie Clustering-Algorithmen. Aus den Clustern leiten sie jedoch zusätzlich ein Modell mit harten oder weichen Grenzen zwischen den einzelnen Gruppen ab. Das zu klassifizierende Dokument wird den Clustern zugeordnet, in deren Grenzen es liegt. Werden einem Text-Corpus häufiger neue Dokumente hinzugefügt oder alte daraus entfernt, können sich Clustergrenzen verschieben – dann sollte das Modell neu berechnet werden.

Sehen ist Verstehen

Die Cluster-Analyse gruppiert Dokumente thematisch, alle Dokumente eines Clusters kreisen um das gleiche Thema. Die absolute Anzahl der Dokumente im Text-Corpus bleibt aber gleich. Das Grundproblem – zu viele Dokumente, um alle zu lesen – bleibt. Um einen Eindruck vom Inhalt des Text-Corpus zu bekommen, gibt es verschiedene Optionen.

Über kleinere Cluster oder auffällige Dokumente, sogenannte Outlier, verschafft man sich einzeln Überblick. Dafür betrachtet man Dokumente als Stichprobe oder liest sie quer. Worum es in dem Cluster geht, lässt sich danach gut einschätzen.

In größeren Clustern sucht man gute Repräsentanten. Man kann entweder das Zentrum des Clusters betrachten und erfährt das zentrale Thema. Oder man verschafft sich Überblick über dessen Bandbreite, repräsentiert in mehreren Dokumenten.

Einen ersten Eindruck über große Cluster bekommt man mit Wordclouds. Sie helfen, das Ergebnis zu verstehen. Eine Wordcloud übernimmt die wichtigsten Schlagworte in eine Grafik und setzt sie durch Größe und Farbe zueinander ins Verhältnis. Das hilft übrigens auch beim Erstellen von Blacklists: Neben dem Wort „Kaffeemaschine“ schließt man mit Blick auf die Wordcloud in Abbildung 2 potenziell auch das Wort „Maschine“ aus, da es keinen Informationsmehrwert bringt (s. Abb. 3)

Von Worten zu Themen – für jedes Projekt neu

Text-Mining bereitet Textinformationen als Vektoren auf und clustert sie. Nach dem ersten Durchlauf lohnt es sich, die Ergebnisse genau zu prüfen und die Schritte zur Datenbereinigung anzupassen – oder das Datenset mit anderen Algorithmen auszuwerten. Mit dem iterativen Vorgehen werden Fehlauswertungen oder falsche Interpretationen weniger wahrscheinlich.

Bei Nachfolgeprojekten ist man verführt, das Bereinigen der Daten abzukürzen und beispielsweise Blacklists aus einem älteren Projekt zu verwenden, oder Häufigkeiten immer genauso zu zählen wie im letzten Projekt. Daraus können falsche Ergebnisse entstehen, weil die Datenbasis für die Auswertungen nicht korrekt war. Gerade bei großen Datenmengen fällt das nicht auf. Man erkennt nicht, wie sich Einschränkungen auf das Datenset auswirken. Wichtig ist es deswegen, jedes Projekt neu aufzusetzen und keine Abkürzungen zu nehmen – egal wie verlockend die Zeitersparnis auch zu sein scheint.

Dr. Lisa Wagner ist seit 2015 Senior IT Consultant bei MaibornWolff. Schon im Studium der Mathematik und Informatik an der RWTH Aachen beschäftigte sie sich mit Data Analysis, Algorithmik und Optimierung und sammelte erste Erfahrungen in der Softwareentwicklung. Beides führt sie bei ihrem jetzigen Arbeitgeber MaibornWolff weiter.
E-Mail: lisa.wagner@maibornwolff.de

JavaSPEKTRUM ist eine Fachpublikation des Verlags:
SIGS DATACOM GmbH
Lindlaustraße 2c · 53842 Troisdorf
Tel.: 0 22 41 / 23 41-100 · Fax: 0 22 41 / 23 41-199
E-Mail: info@sigs-datacom.de · www.javaspektrum.de
SIGS DATACOM
FACHINFORMATIONEN FÜR IT-PROFESSIONALS