IM SOG DER DATEN

Big Data Analytics mit Revolution R

Mit Big Data ist in den letzten Jahren ein Buzzword entstanden. Es beschreibt die Analyse von Datensätzen, die mit gängigen Methoden nicht mehr analysierbar sind. Mit zunehmend günstigerem Festplattenplatz haben Unternehmen begonnen, alles zu speichern, was verwertet werden kann: Verkaufsdaten, Tweets, GPS-Handydaten, Verkehrsdaten, und vieles mehr. Schon 2013 wurde berichtet, dass 90 % aller auf der Welt verfügbaren Daten in den letzten beiden Jahren entstanden sind. Seit 2013 hat sich die Größe des globalen Datenmeers wohl noch einmal verzehnfacht. Einige Unternehmen sammeln und analysieren inzwischen Datenmengen im Petabyte-Bereich: man denke zum Beispiel an Amazon, Facebook, und LinkedIn.

Das Auswerten dieser oft unstrukturierten Daten öffnet die Türen für viele neue Vorhersagemethoden: Kaufempfehlungen basierend auf vorher angesehenen und gekauften Artikeln, Betrugserkennung, oder die Analyse von Marketingkampagnen sind nur einige der Bereiche, die enorme Vorteile aus Big-Data-Analysen ziehen. Eine von IBM in Auftrag gegebene Studie kam 2012 zu dem folgenden Schluss: Der Return on Investment beträgt bei Business-Analytics-Lösungen etwa 250% - wenn sie vorhersagende Analysen anstellen. [1]

Die Vorteile, die die Analyse dieser Datenmasse mit sich bringt, sind klar. Es hat sich jedoch herausgestellt, dass es bedeutend schwieriger ist diese Daten zu verwalten und sie zu analysieren. Der limitierende Faktor ist hier meist entweder der Arbeitsspeicher oder die Rechenleistung. Diese Datensätze sind zum einen zu groß um komplett in den Arbeitsspeicher geladen zu werden, und zum anderen sind etablierte Programme zu langsam, um sie dann mit konventionellen Methoden zu analysieren.

Wie bändigt man nun diese Daten? In den letzten Jahren sind hierfür eine Vielzahl von Umgebungen entwickelt worden. Die verbreitetste ist wohl Apache Hadoop, eine Kombination des verteilten Dateisystems Hadoop Distributed File System (HDFS) zum Speichern von Datensätzen, die nicht mehr auf eine Festplatte passen, und der MapReduce-Algorithmus, mit dem diese verteilten Datensätze analysiert werden können. Eine weitere Möglichkeit, die am Ende dieses Artikels in einem Use Case vorgestellt wird, ist eine Distribution der Programmiersprache R von der jungen Microsoft-Tochter Revolution Analytics.

NOT SO BIG DATA

Hadoop ist zwar ein Framework, das mit beliebig großen Datensätzen umgehen kann, aber es bringt doch eine zusätzliche Schicht an Komplexität mit in die Analyse. Außerdem ist man mit Hadoop beschränkt auf das MapReduce-Paradigma, in dem man seine Berechnungen implementieren muss. Man sollte also nur auf ein verteiltes System zurückgreifen, wenn die Grenzen eines einzelnen Servers wirklich überschritten sind.

Das Implementieren und Einsetzen einer Hadoop-Umgebung zur Analyse eines 1TB-Datensatzes klingt zwar hip, ist aber mit sehr hoher Wahrscheinlichkeit mit einem unnötigen Kosten- und Zeitaufwand verbunden. Daten, die noch auf einen einzigen Rechner oder Server passen, analysiert man besser mit flexibleren Methoden – bei weniger als 4TB ist das definitiv der Fall. Eine dieser Methoden ist die Open Source-Programmiersprache R, die speziell für

statistische Berechnungen und Visualisierungen gemacht ist.

R: DIE PROGRAMMIERSPRACHE FÜR DATENANALYSE

GNU R ist eine Programmiersprache, die speziell für statistisches Arbeiten konzipiert ist. Die freie Software existiert seit den 90ern und ist eine Umgebung geworden, mit der man Daten aus allerlei Quellen importieren, analysieren und visualisieren kann. R ist inzwischen in Wissenschaft und Industrie gleichermaßen verbreitet, und findet Anwendung in jeglichen Gebieten wo Data Science anfällt - von der Geologie bis zur Finanzbranche.

Eine besondere Stärke von R sind die Tausenden von Paketen, die von Benutzern erstellt werden und über das Comprehensive R Archive Network (CRAN) frei verteilt werden. Dadurch ist es möglich, zum Beispiel hochmoderne Data Mining-Algorithmen in kürzester Zeit in R anwenden zu können, statt auf ein nächstes Release warten zu müssen.

Es existieren Pakete mit Anbindungen zu SQL-Datenbanken (zum Beispiel RMySQL, RPostgreSQL, ROracle) und zu anderen Programmiersprachen (so wie Rcpp, rPython und rJava). Andere Pakete können zum Beispiel auf Tweets (twitteR) oder Facebook-Beiträge (Rfacebook) zugreifen, oder Daten aus Google Analytics (RGoogleAnalytics) importieren.

Mit der RStudio-GUI gibt es nun auch eine praktische grafische Umgebung, die Codedateien, Konsolenoutput, und Grafikfenster komfortabel vereint.

R ist eine moderne Programmiersprache: deshalb ist sie auf Kaggle, einer Plattform für Data-Science-Wettbewerbe auch die meistverwendete Sprache[2].

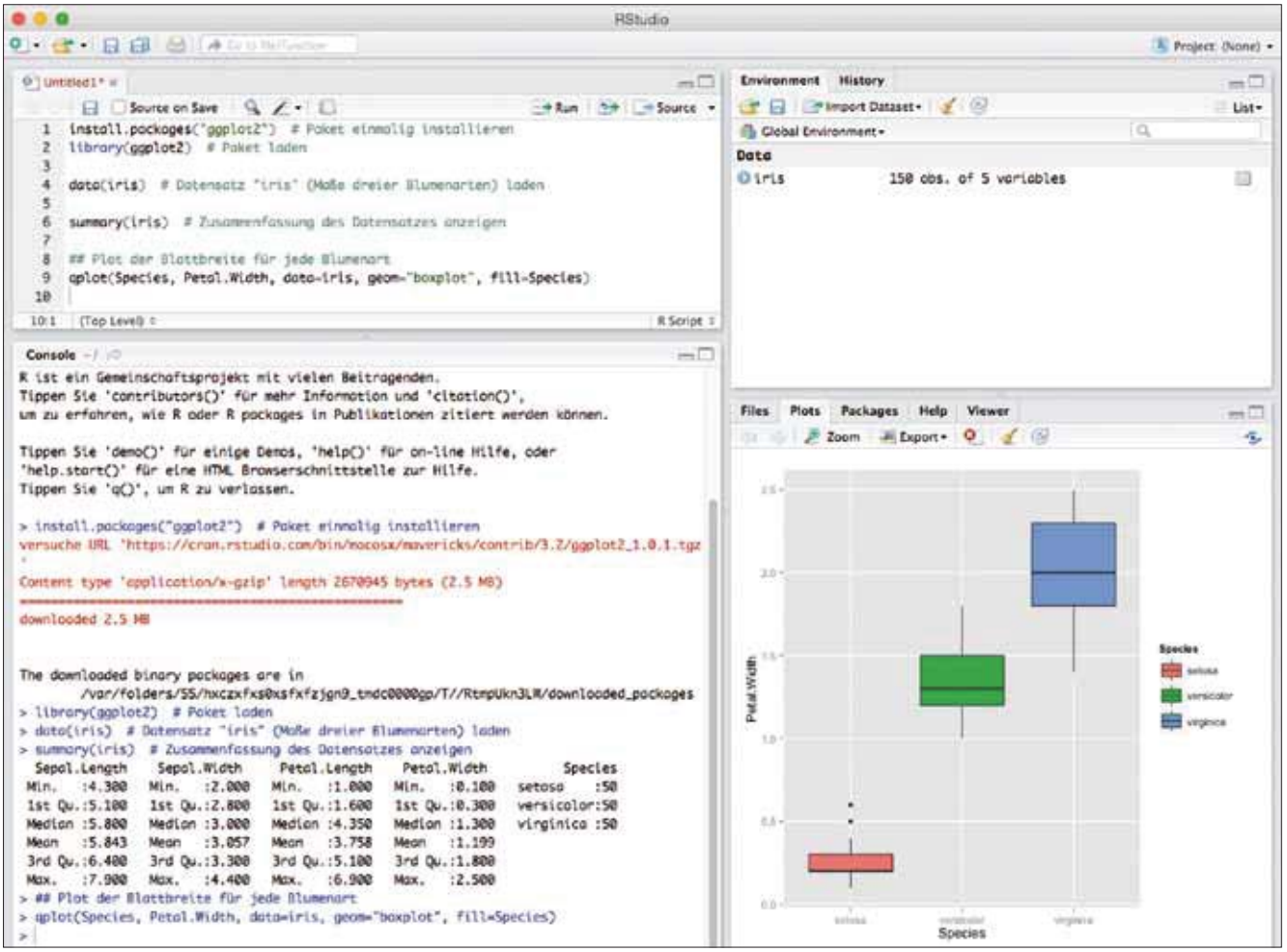
Ein großer Nachteil von R ist, dass es in der Standardausführung nicht optimiert auf Geschwindigkeit und Speicherverbrauch ist. Es kann von Natur aus nur mit Datensätzen arbeiten, die es komplett in den Arbeitsspeicher laden kann. Dazu kommt, dass bei Funktionsaufrufen Datensätze oft implizit kopiert werden, womit der Datensatz nun den doppelten Speicher belegt. Was vor einigen Jahren noch unproblematisch war, stellte sich mit wachsenden Datenmengen als großes Problem dar.

Als Gegenmaßnahme sind mit der Zeit eine Reihe an R-Paketen entstanden, die zum Beispiel mit Daten arbeiten können, die größer als der verfügbare Arbeitsspeicher sind, und explizite Parallelisierung implementieren können. Diese Pakete sind allerdings oft nur für die Lösung eines speziellen Problems gedacht. Dann arbeiten sie nicht gut zusammen.

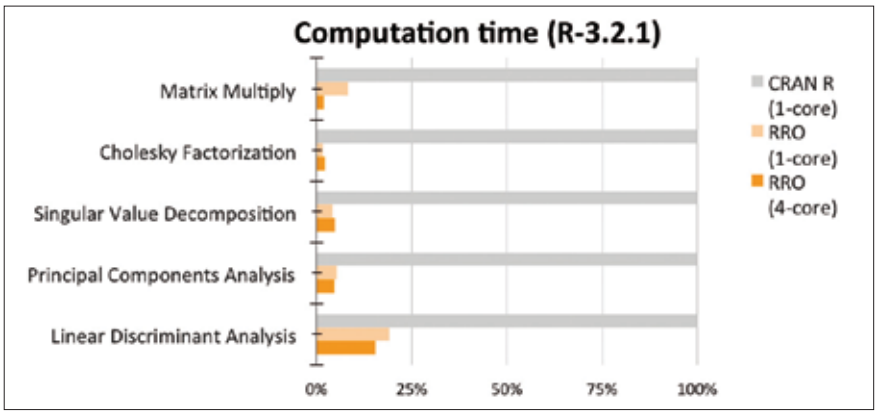
Im Jahr 2007 erschien eine kommerzielle R-Distribution, Revolution R, die eine einheitliche Lösung für alle Einschränkungen von GNU R bereitstellt.

REVOLUTION R: DIE BRÜCKE ZWISCHEN R UND BIG DATA

Die amerikanische Firma Revolution Analytics machte es sich zur Aufgabe, eine kompatible R-Distribution



RStudio, die verbreitetste IDE für R mit Skriptfenster, Konsole und Fenster für vorhandene Variablen und Datensätze sowie Plots.



Der Vergleich von Laufzeiten für häufige, rechenaufwändige Operationen (Zeiten für die Standarddistribution GNU R (graue Balken), und das ebenso frei verfügbare Revolution R Open)[3].

zu veröffentlichen, die zum einen schnellere Algorithmen für häufig vorkommende Berechnungen bereitstellt, und zum anderen mit Big Data arbeiten kann. Dass dieses Konzept aufging, zeigt wohl am besten die Tatsache, dass Revolution Analytics kürzlich von Microsoft aufgekauft wurde.

Sie bieten zwei Versionen ihrer Software an: Das freie, kostenlose Revolution R Open (RRO), und das kostenpflichtige, mächtige Revolution R Enterprise (RRE).

Das freie RRO kommt bereits mit der Intel Math Kernel Library, das implizites Multithreading bei vielen mathematischen Operationen erlaubt, und selbst bei Singlecore-Operationen teilweise 10- bis 40-mal schneller ist als das herkömmliche R.

Mit RRE erhält man zusätzlich zu den Vorteilen von RRO noch kommerziellen Support, und insbesondere eine Reihe von proprietären R-Paketen, wobei das wichtigste wohl RevoScaleR ist. Dieses Paket liefert eine Reihe von Funktionen, mit denen die Analyse von Datensätzen im Terabyte-Bereich ermöglicht wird, ohne auf teure, spezialisierte Hardware zurückgreifen zu müssen.

RevoScaleR führt ein eigenes Dateiformat .XDF (eXternal Data Frame) ein, das Datensätze platzsparend in einem binären Format speichert. Dieses Format erlaubt es auch, sequenziell auf einzelne Reihen oder Spalten des Datensatzes zuzugreifen, ohne die ganze Tabelle in den Speicher laden zu müssen.

Zusätzlich zum platzsparenden Speichern von Datensätzen bringt RevoScaleR Funktionen zum Import, zur Manipulation, Analyse und Visualisierung mit, die automatisch von Multicore-Umgebungen Gebrauch machen. Der Unterschied zu RRO ist hier, dass in RRO nur die low-level Operationen (zum Beispiel Matrixinversionen) beschleunigt wurden, und dass in RRE auch Regressionsmodelle und Grafiken auf XDF-Daten angewendet werden können. Die RevoScaleR-Funktionen lesen hierzu nacheinander Blöcke der Datensätze ein - auf diese Weise sind nie die gesamten Daten im Arbeitsspeicher. Das Paradigma ist hier ganz ähnlich zu Hadoops MapReduce-Algorithmus: Daten blockweise lesen und auswerten, Zwischenergebnisse speichern, und diese am Ende zusammenfassen. Der Vorteil an RRE ist natürlich, dass alles bereits implementiert ist, und der Benutzer nur noch einen Funktionsaufruf tätigt. Diese Vorgehensweise wird Parallel External Memory Algorithm (PEMA) genannt, da die meisten Daten außerhalb des Arbeitsspeichers sind, und die Stücke parallel verarbeitet werden.

Das Verteilen auf mehrere Rechner in einer Clusterumgebung funktioniert übrigens denkbar einfach, nämlich mit einer zusätzlichen Zeile R-Code vor den Berechnungen.

USE CASE: FLUGVERSPÄTUNGEN

Ein schöner Beispieldatensatz ist der Airlines-Datensatz, in dem über 120 Millionen amerikanische Inlandsflüge gespeichert sind. Die festgehaltenen Variablen beinhalten die Start- und Landezeit, die Verspätungen, die Start- und Zielflughäfen, und die geflogenen Meilen.

Man kann sich den Datensatz unter [4] selbst downloaden.

Mit diesen Daten kann man sich gründlich austoben, und unzählige Fragestellungen überprüfen. In diesem Beispiel sehen wir uns an, wie die Verspätung beim Abflug mit der durchschnittlichen Geschwindigkeit während des Fluges zusammenhängen - ob also Flüge, die verspätet starten, dafür schneller fliegen.

Im ersten Schritt lädt man dazu die

```
## Declare the file paths for the csv and xdf files
myAirlineCsv <- file.path(rxGetOption("sampleDataDir"), "2007_subset.csv")
myAirlineXdf <- "2007_subset.xdf"

## Use rxImport to import the data into xdf format
rxImport(inData = myAirlineCsv, outFile = myAirlineXdf, overwrite = TRUE)

## Get basic information about your data
rxGetInfo(data = myAirlineXdf,
          getVarInfo = TRUE,
          numRows = 4)


## File name: /tmp/Rserv/conn12238/2007_subset.xdf
## Number of observations: 149139
## Number of variables: 5
## Number of blocks: 1
## Compression type: zlib
## Variable information:
## Var 1: ActualElapsedTime, Type: integer, Low/High: (17, 730)
## Var 2: AirTime, Type: integer, Low/High: (0, 704)
## Var 3: Distance, Type: integer, Low/High: (31, 4962)
## Var 4: DepDelay, Type: integer, Low/High: (-38, 1243)
## Var 5: ArrDelay, Type: integer, Low/High: (-63, 1262)
## Data (10 rows starting with row 1):
##    ActualElapsedTime AirTime Distance DepDelay ArrDelay
## 1                 62      44      237       24       26
## 2                191     179     1262       -3      -27
## 3                138     127     1111       24        2
## 4                131     110      842       18       19
```

```
## Calculate an additional variable: airspeed
##     (distance traveled / time in hours in the air).
rxDataStep(inData = myAirlineXdf,
           outFile = myAirlineXdf,
           varsToKeep = c("AirTime", "Distance"),
           transforms = list(airSpeed = Distance / AirTime * 60 ),
           append = "cols",
           overwrite = TRUE)

## Correlation for departure delay, arrival delay, and air speed
rxCor(formula = ~ DepDelay + ArrDelay + airSpeed,
      data = myAirlineXdf,
      rowSelection = (airSpeed > 50) & (airSpeed < 800))


##             DepDelay    ArrDelay    airSpeed
## DepDelay 1.00000000  0.93157838  0.02196339
## ArrDelay 0.93157838  1.00000000 -0.06668119
## airSpeed 0.02196339 -0.06668119  1.00000000
```

CSV-Datei in R ein. Da eine Teilmenge dieses Datensatzes als Beispieldaten in RRE vorhanden ist, laden wir sie aus dem Beispieldatenverzeichnis. Im Anschluss wird die CSV in eine effizientere XDF-Datei umgewandelt und gespeichert. Mit der Funktion rxGetInfo() können wir uns nun eine Zusammenfassung des Datensatzes ansehen.

Da die durchschnittliche Fluggeschwindigkeit noch nicht als Variable gespeichert ist, erstellen wir sie, indem wir die geflogene Distanz in Meilen durch die Flugzeit in Stunden dividieren. Im nächsten Schritt berechnen wir die Korrelationsmatrix der drei Variablen „Verspätung bei Start“, „Verspätung bei Landung“, und „durchschnittliche Fluggeschwindigkeit“.

Dieser Schritt beinhaltet übrigens schon eine Matrixmultiplikation, was durch Revolution R enorm beschleunigt wird.

Als letztes berechnen wir mit der Funktion rxLinMod() ein lineares Regressionsmodell, mit der Zielgröße „Geschwindigkeit“ und der Einflussgröße „Verspätung bei Start“. Mit dem Argument rowSelection beschränken wir den Datensatz auf die Beobachtungen, in denen die durchschnittliche Geschwindigkeit über 50 und unter 800 Meilen pro Stunde liegt. Dies dient dazu, klare Ausreißer und fehlerhafte Daten auszuschließen.

Der resultierende Schätzer für DepDelay, also die Verspätung bei Abflug, beträgt 0.04638. Das bedeutet, dass für jede Minute Verspätung die durchschnittliche Fluggeschwindigkeit um 0,046 Meilen pro Stunde schneller wird. Eine Verspätung von 60 Minuten resultiert also in einer um 2,8 Meilen pro Stunde schnelleren Fluggeschwindigkeit. Auch wenn dieser Effekt statistisch signifikant ist, ist er von der Größe her doch zu vernachlässigen. Wir konnten also keinen nennenswerten Hinweis darauf feststellen, dass verspätete Flüge schneller fliegen um die verlorene Zeit wieder aufzuholen.

```
## Regression for airSpeed based on departure delay
myLMobj <- rxLinMod(formula = airSpeed ~ DepDelay,
                    data = myAirlineXdf,
                    rowSelection = (airSpeed > 50) & (airSpeed < 800))
summary(myLMobj)


## Call:
## rxLinMod(formula = airSpeed ~ DepDelay, data = myAirlineXdf,
##     rowSelection = (airSpeed > 50) & (airSpeed < 800))
##
## Linear Regression Results for: airSpeed ~ DepDelay
## Data: myAirlineXdf (RxXdfData Data Source)
## File name: 2007_subset.xdf
## Dependent variable(s): airSpeed
## Total independent variables: 2
## Number of valid observations: 145054
## Number of missing observations: 0
##
## Coefficients:
##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept) 3.901e+02  2.091e-01 1866.095 2.22e-16 ***
## DepDelay    4.638e-02  5.543e-03    8.367 2.22e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 75.95 on 145052 degrees of freedom
## Multiple R-squared: 0.0004824
## Adjusted R-squared: 0.0004755
## F-statistic: 70.01 on 1 and 145052 DF,  p-value: < 2.2e-16
## Condition number: 1
```

LINKS

[1] http://www.besmart.company/MKT/Promos/2012/0612_PA/0612_businessvalue_PA.pdf

[2] https://www.kaggle.com/

[3] https://mran.revolutionanalytics.com/documents/rro/multithread/#mt-bench

[4] http://stat-computing.org/dataexpo/2009/the-data.html

ALEXANDER ENGELHARDT ist Statistiker mit besonderem Interesse am Machine Learning. Er unterstützt als Freelancer Unternehmen bei Datenauswertungen und dem Erstellen von Prognosemodellen.

engelhardt@alpha-epsilon.de

http://www.alpha-epsilon.de/