KÜNSTLICHE NEURONALE NETZWERKE IN SQL SERVER EINSETZEN

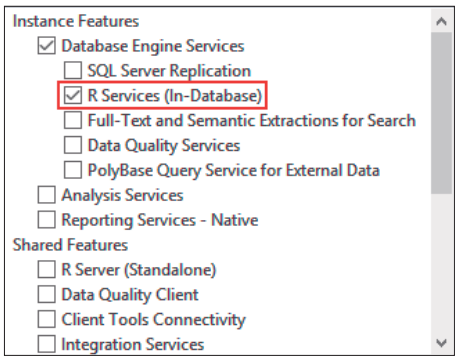
Smarte Datenbank

Mit MicrosoftML und SQL Server R Services schafft Microsoft die Möglichkeit, Machine-Learning-Algorithmen nahe an der Datenquelle zu nutzen.

Artificial Intelligence (AI), zu Deutsch künstliche Intelligenz (KI), ist wieder in aller Munde [1], denn seit einigen Jahren feiern namhafte Softwarehersteller, darunter neben Google auch Microsoft, im Bereich des maschinellen Lernens (ML) und insbesondere im Deep Learning außergewöhnliche Erfolge [2].

Deep-Learning-Methoden basieren auf künstlichen neuronalen Netzwerken mit mehreren – sogenannten verborgenen – Schichten zwischen Eingabe- und Ausgabeschicht. Durch geeignete Wahl der Schichten und deren Verknüpfung ist die Auswertung großer Mengen komplexer oder ungenauer Eingangsdaten in einer Qualität möglich, die bisher dem menschlichen Gehirn vorbehalten beziehungsweise überhaupt noch nicht realisierbar war.

Typische Anwendungsszenarien sind die Bild- und Spracherkennung, semantische Textanalyse, Voraussage von zeitlichen Entwicklungen, Bildung von Handlungsstrategien und vieles mehr [3]. Zuletzt erlangte AlphaGo Berühmtheit, als es spektakulär den Weltmeister im Go-Spiel schlug [4]. Zwei kurze, aber gut gemachte Einführungen in neuronale Netzwerke und ihre gegenwärtigen Fähigkeiten finden Sie unter [5] und [6].



Bei der Installation von SQL Server muss das Feature „R Services (In-Database)“ aktiviert werden (Bild 1)

Microsoft R Us

Die Zeichen der Zeit hat auch Microsoft erkannt und mit MicrosoftML [7] eine Komponente geschaffen, mit der Entwickler neuronale Netze mit geringem Aufwand erzeugen und in eigenen Softwareprojekten einsetzen können. Dabei kommt das Framework R [8] zur Anwendung, das sich für statistische Berechnungen allgemein durchgesetzt hat.

MicrosoftML setzt als Laufzeitumgebung Microsoft R Open 3.3.2 [9] voraus. Sowohl MicrosoftML als auch Microsoft R Open sind in den folgenden Produkten enthalten:

- Microsoft R Client 3.3.2 [10],
- Microsoft R Server [11],
- Microsoft SQL Server vNext mit SQL Server R Services [12].

Tabelle 1 beschreibt die Produktfamilie Microsoft R. Die Produkte wurden 2015 durch Übernahme des Unternehmens Revolution Analytics eingekauft, weshalb man gelegentlich noch auf das Kürzel *Revo* stößt, wie zum Beispiel im Paketnamen *RevoScaleR*.

Üblicherweise wird ein neuronales Netz zunächst auf einem Arbeitsplatzrechner erstellt. Dort kann es auch trainiert und getestet werden. Bei zufriedenstellender Qualität wird es anschließend für den Produktiveinsatz auf einem leistungsfähigen Server bereitgestellt. Sofern bereits ein SQL Server im Einsatz ist, bietet sich dieser als Server geradezu an, denn hier kann die Datenverarbeitung nahe an der Datenquelle erfolgen. Das neuronale Netz wird dabei in einer Tabelle im Binärformat abgelegt und bei Bedarf per R-Skript geladen. Die Kommunikation zwischen dem Dienst des SQL Servers und den R Services ist dabei verschlüsselt, was dem Datenschutz zugutekommt.

Beachten Sie aber, dass erst die Version vNext des SQL Servers alle erforderlichen Komponenten mitbringt – das ist die Version,

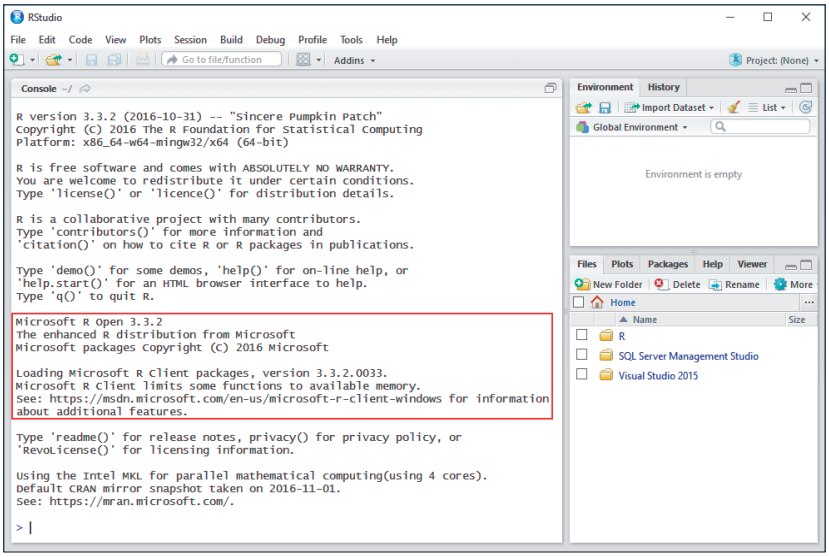
Tabelle 1: Microsoft R – die Produkte

Produkt	Beschreibung	Kostenlos	Quelloffen
Microsoft R Open	Das Standard-R, das von Microsoft um einige Funktionen erweitert wurde, insbesondere zur Nutzung von Mehrkernprozessoren.	Ja	Ja
Microsoft R Client	Arbeitsplatzumgebung für die Entwicklung von R-Skripten. Beinhaltet unter anderem Microsoft R Open, MicrosoftML und ScaleR (Komponenten zur Unterstützung paralleler Verarbeitung, Verbesserung der Geschwindigkeit und Konnektivität zu Datenquellen wie SQL Server und Hadoop).	Ja	Nein
Microsoft R Server	Die kommerzielle Umgebung für hochskalierbare R-Anwendungen im Produktivbetrieb.	Nein	Nein
SQL Server R Services	Erweiterung des Datenbankmoduls von SQL Server zur Ausführung von R-Skripten.	Ja, als Komponente von SQL Server	Nein

die auf SQL Server 2016 folgt. SQL Server vNext befand sich, als dieser Artikel geschrieben wurde, noch in der Preview-Phase und sollte nicht produktiv eingesetzt werden. Am besten installieren Sie ihn in einer virtuellen Maschine, da Microsoft keine saubere Deinstallation garantiert.

Das Beispielprojekt

Dieser Artikel beschreibt die notwendigen Schritte, um ein einfaches neuronales Netzwerk zu erstellen und in SQL Server einzusetzen. Dazu werden Beispieldaten herangezogen, die aus dem frei verfügbaren Paket MASS (Modern Applied Statistics with S [13]) stammen; es handelt sich um Gewebebiopsie-Daten, die aus der Untersuchung des Tumorgewebes von Brustkrebspatientinnen gewonnen wurden. Nach dem Training soll das neuronale Netz in der Lage sein, anhand neuer Beobachtungen eine Diagnose zu stellen, das heißt gutartiges (englisch benign) von bösartigem (malignant) Gewebe zu unterscheiden. Es erfolgt also eine Einteilung der Tumore in zwei Klassen (binäre Klassifikation).



Die Arbeitsfläche von RStudio mit geladenem Microsoft R Open 3.3.2 (Bild 2)

Als Entwicklungsumgebung (IDE) soll Microsoft R Client kombiniert mit RStudio [14] zum Einsatz kommen. Microsoft hat mit den R Tools for Visual Studio [15] seit Kurzem auch eine eigene IDE am Start; diese reicht aber an den Reifegrad von RStudio derzeit nicht heran.

Installation von SQL Server mit R Services

Installieren Sie zunächst den SQL Server in der aktuellen Version, die Sie unter [16] laden können. In diesem Artikel wurde SQL Server vNext CTP 1.4 eingesetzt. Wählen Sie bei der Installation die Developer Edition und stellen Sie sicher, dass *R Services (In-Database)* aktiviert ist, vergleiche Bild 1. Danach müssen Sie noch die Lizenzbedingungen von Microsoft R Open akzeptieren.

Damit Sie mit SQL Server arbeiten können, installieren Sie außerdem das SQL Server Management Studio. Auch hier müssen Sie die neueste Version wählen (17.0 RC1 oder höher). Den Download-Link finden Sie unter [17].

Um R-Code in SQL Server nutzen zu können, muss zunächst die Ausführung externer Skripts erlaubt werden. Dies klappt mit folgenden SQL-Befehlen:

```
EXEC sp_configure 'external scripts enabled', 1
RECONFIGURE WITH OVERRIDE
```

Starten Sie die SQL-Server-Instanz anschließend neu und stellen Sie mit folgendem SQL-Skript sicher, dass die Aktivierung erfolgreich war:

```
EXEC sp_execute_external_script
  @language = N'R',
  @script = N'OutputDataSet <- InputDataSet',
  @input_data_1 = N'SELECT 1 AS Test'
WITH RESULT SETS ((Test bit))
```

Als Ergebnis auf diese Eingaben sollten Sie eine einzelne Zeile mit dem Wert *1* erhalten. Legen Sie für die späteren Aufgaben außerdem noch eine einfache Datenbank an, wie das in Listing 1 gezeigt wird.

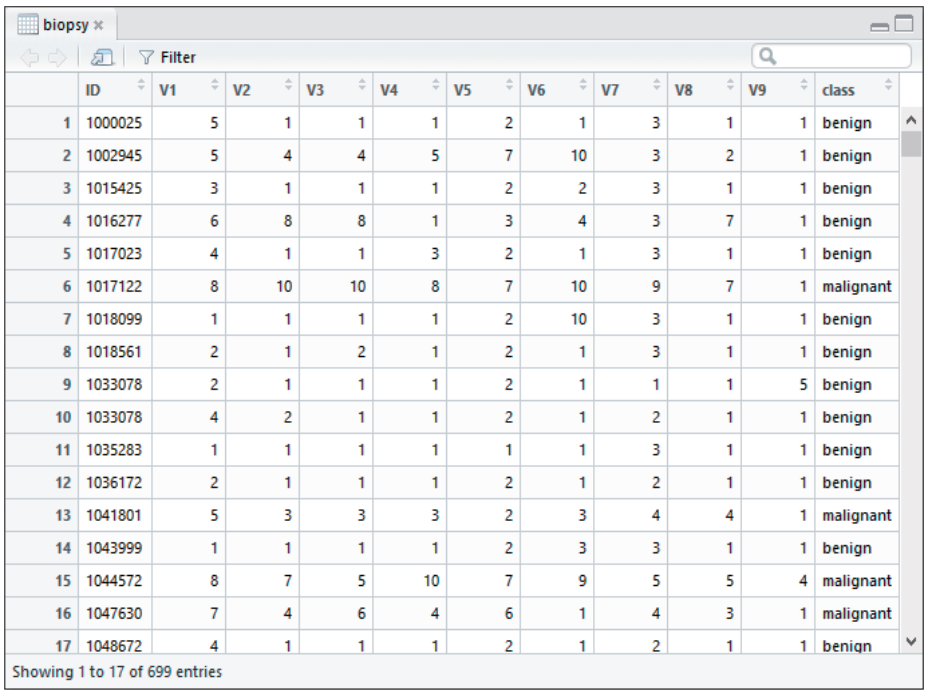
Listing 1: Demo-Datenbank im Standardpfad

```
CREATE DATABASE NeuralNetDemo
ON PRIMARY
(
  NAME = N'NeuralNetDemo',
  FILENAME = N'C:\Program Files\
              Microsoft SQL Server\
              MSSQL14.MSSQLSERVER\MSSQL\DATA\
              NeuralNetDemo.mdf'
)

LOG ON
(
  NAME = N'NeuralNetDemo_log',
  FILENAME = N'C:\Program Files\
              Microsoft SQL Server\
              MSSQL14.MSSQLSERVER\MSSQL\DATA\
              NeuralNetDemo_log.ldf'
)
```

Installation der IDE

Setzen Sie im nächsten Schritt die Entwicklungsumgebung auf. Installieren Sie Microsoft R Client, der unter [18] zur Verfügung steht. Per Standardeinstellung sammelt der R Client Daten über Ihr Nutzerverhalten. Falls das bei Ihnen Unbehagen auslöst, können Sie die Funktion abstellen, indem Sie ▶



biopsy

Filter

	ID	V1	V2	V3	V4	V5	V6	V7	V8	V9	class
1	1000025	5	1	1	1	2	1	3	1	1	benign
2	1002945	5	4	4	5	7	10	3	2	1	benign
3	1015425	3	1	1	1	2	2	3	1	1	benign
4	1016277	6	8	8	1	3	4	3	7	1	benign
5	1017023	4	1	1	3	2	1	3	1	1	benign
6	1017122	8	10	10	8	7	10	9	7	1	malignant
7	1018099	1	1	1	1	2	10	3	1	1	benign
8	1018561	2	1	2	1	2	1	3	1	1	benign
9	1033078	2	1	1	1	2	1	1	1	5	benign
10	1033078	4	2	1	1	2	1	2	1	1	benign
11	1035283	1	1	1	1	1	1	3	1	1	benign
12	1036172	2	1	1	1	2	1	2	1	1	benign
13	1041801	5	3	3	3	2	3	4	4	1	malignant
14	1043999	1	1	1	1	2	3	3	1	1	benign
15	1044572	8	7	5	10	7	9	5	5	4	malignant
16	1047630	7	4	6	4	6	1	4	3	1	malignant
17	1048672	4	1	1	1	2	1	2	1	1	benign

Showing 1 to 17 of 699 entries

Einige Datensätze aus dem Beispiel „biopsy“ (Bild 3)

die mitgelieferte R Console – Sie finden die Console üblicherweise im Pfad *C:\Program Files\Microsoft\R Client\R_SERVER\bin\x64\RGui.exe* – mit Administratorrechten starten und anschließend den folgenden Befehl ausführen:

```
rxPrivacyControl (FALSE)
```

Installieren Sie zudem noch das Hilfsprogramm RStudio Desktop [19]. Nach dem Start sollte sich Microsoft R Open dann melden, siehe Bild 2, andernfalls tragen Sie im Menü unter *Tools* | *Global Options* | *Registerkarte „General“* | *R Version* den Pfad zum R Client ein – der Standardpfad ist *C:\Program Files\Microsoft\R Client\R_SERVER*.

Definition des Beispielmodells

Nun kann der spannende Teil in RStudio beginnen. Laden Sie mit dem Befehl *library(MASS)* das Paket mit den Beispieldaten. Mithilfe von *View(biopsy)* erhalten Sie einen Überblick über die Datenstruktur (Bild 3).

Tabelle 2 erläutert die Bedeutung der Felder; *V1* bis *V9* bezeichnen die gemessenen Laborwerte, die sich auf einer Skala von 1 bis 10 bewegen – je höher die in *V1* bis *V9* angegebene Zahl, desto stärker trifft das jeweilige Merkmal zu.

Das Feld *class* enthält die Diagnose; sie lautet entweder *benign* oder *malignant*. Einen dieser beiden Werte soll das neuronale Netz später anhand von neuen Messdaten voraussagen.

Zur Definition neuronaler Netze hat Microsoft die Sprache Net# ins Leben gerufen, die unter [20] im Detail erklärt wird. Mit ihrer Hilfe lässt sich deklarativ festlegen, welche Schichten das Netzwerk und wie viele Neuronen jede Schicht bekommt, welche Verbindungen zwischen den Schichten bestehen sollen und einiges mehr. Für Demonstrationszwecke reicht die folgende einfache Netzwerkarchitektur aus:

```
net <- ("
  input In auto;
```

Tabelle 2: Bedeutung der Datenfelder aus Bild 3

Feldname	Datentyp	Bedeutung
ID	integer	Nummer der Gewebeprobe
V1	integer [1 –10]	Verklumpungsstärke
V2	integer [1 –10]	Einheitlichkeit der Zellgröße
V3	integer [1 –10]	Einheitlichkeit der Zellform
V4	integer [1 –10]	Verklebung der Außenränder
V5	integer [1 –10]	Größe einzelner Epithelzellen
V6	integer [1 –10]	Häufigkeit freiliegender Zellkerne
V7	integer [1 –10]	Farbloses Chromatin
V8	integer [1 –10]	Normale Zellkernkörperchen
V9	integer [1 –10]	Häufigkeit an Mitosen
class	character	Diagnose

Listing 2: Trainings- und Testmenge erzeugen

```
# Mengengröße festlegen
sample_size <- floor(.85 * nrow(biopsy))

# Seed setzen, damit die Mengen reproduzierbar
# erzeugt werden können
set.seed(13)

# Indices für die Trainingsmenge auswählen
train_indexes <- sample(
  seq_len(nrow(biopsy)),
  size = sample_size
)

# Trainings- und Testmenge erzeugen
train_data <- biopsy[train_indexes,]
test_data <- biopsy[-train_indexes,]
```

	ID	class	V1	V2	V3	V4	V5	V6	V7	V8	V9	PredictedLabel	Score.malignant	Probability.malignant
1	1033078	benign	2	1	1	1	2	1	1	1	5	benign	-2.7587433	0.05959466
2	1066979	benign	5	1	1	1	2	1	2	1	1	benign	-2.4397941	0.08018821
3	1079304	benign	2	1	1	1	2	1	2	1	1	benign	-2.8236048	0.05606204
4	1115282	malignant	5	3	5	5	3	3	4	10	1	malignant	0.5872537	0.64273524
5	1116192	benign	1	1	1	1	2	1	2	1	1	benign	-2.9518006	0.04965136
6	1136142	benign	2	1	1	1	3	1	2	1	1	benign	-2.7341237	0.06098940
7	1137156	benign	2	2	2	1	1	1	7	1	1	benign	-2.0994301	0.10915245
8	1148873	malignant	3	6	6	6	5	10	6	8	3	malignant	2.4330401	0.91931224
9	1152331	benign	4	1	1	1	2	1	3	1	1	benign	-2.4616854	0.07858838
10	1160476	benign	2	1	1	1	2	1	3	1	1	benign	-2.7174764	0.06194977
11	1164066	benign	1	1	1	1	2	1	3	1	1	benign	-2.8455765	0.05491057
12	1165790	benign	5	1	1	1	2	1	3	1	1	benign	-2.3340199	0.08834414
13	1166630	malignant	7	5	6	10	5	10	7	9	4	malignant	3.3751435	0.96691871
14	1167471	benign	4	1	2	1	2	1	3	1	1	benign	-2.3192916	0.08953754
15	1180831	benign	3	1	1	1	3	1	2	1	1	benign	-2.6061358	0.06874464
16	1182404	benign	4	1	1	1	2	1	2	1	1	benign	-2.5675874	0.07125397
17	1183240	benign	4	1	2	1	2	1	2	1	1	benign	-2.4250474	0.08128261
18	1197270	benign	3	1	1	1	2	1	3	1	1	benign	-2.5895090	0.06981678
19	1197440	benign	1	1	1	2	1	3	1	1	7	benign	-2.3907928	0.08387744
20	1197979	benign	4	1	1	1	2	2	3	2	1	benign	-2.1438184	0.10491057

Die ersten 20 Zeilen von predict_net (Bild 4)

```
  hidden Hidden [200] sigmoid from In all;
  output Out [1] sigmoid from Hidden all;
")
```

Das so definierte Netz besteht aus einer Eingabeschicht (*input*), einer verborgenen Schicht (*hidden*) und einer Ausgabeschicht (*output*). Die Anzahl an Neuronen in der Eingabeschicht wird dynamisch bei der Zuweisung der Eingangsdaten festgelegt (*auto*).

Die verborgene Schicht erhält 200 Neuronen, die Ausgabeschicht ein einzelnes Neuron.

Alle Neuronen der Eingangsschicht werden mit allen Neuronen der verborgenen Schicht verknüpft (*all*), welche wiederum alle mit dem Neuron der Ausgabeschicht verknüpft werden.

Für die verborgene und die Ausgabeschicht wurde *sigmoid* als Aktivierungsfunktion gewählt.

Listing 3: Grafische Qualitätskontrolle

```
par(mfrow = c(2,1))

with(
  predict_net[predict_net$class == "benign",],
  plot(
    Probability.malignant,
    main = "benign",

    ylim = c(0,1),
    pch = 19,
    col = "green3"
  )
)

abline(h = c(.5), lty = 2)

with(
  predict_net[predict_net$class == "malignant",],
  plot(
    Probability.malignant,
    main = "malignant",

    ylim = c(0,1),
    pch = 19,
    col = "red"
  )
)

abline(h = c(.5), lty = 2)
```

Training und Bewertung

Trennen Sie als Nächstes die vorliegenden Daten in zwei Mengen auf: Die erste Menge (Trainingsmenge) dient zum Trainieren des Netzes, die zweite (Testmenge) zur Bewertung der Qualität. Das Verhältnis soll 85 zu 15 Prozent sein, vergleiche Listing 2.

Nun können Sie das Netz trainieren. Dabei müssen Sie wie folgt das vorauszusagende Feld (*class*), die Eingabefelder (*V1* bis *V9*), die Trainingsmenge (*train_data*) und das zu nutzende Modell (*net*) angeben:

```
trained_net <- rxNeuralNet(
  class ~ V1 + V2 + V3 + V4 + V5 +
          V6 + V7 + V8 + V9,
  data = train_data,
  netDefinition = net
)
```

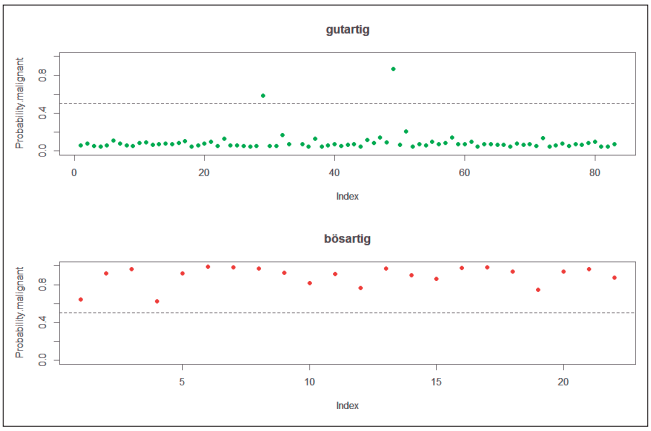
Die Funktion *rxNeuralNet* erlaubt eine Reihe weiterer optionaler Parameter zur Feinsteuerung des Trainings [21]. So können Sie beispielsweise die Rechenkraft der Grafikkarte (GPU) nutzen, wodurch sich das Training bis um den Faktor neun beschleunigen lässt [22].

Trifft das trainierte Modell nun realistische Diagnosen? Das ermitteln Sie, indem Sie es auf die Testmenge ansetzen:

```
predict_net <- rxPredict(
  trained_net,
  test_data,
  writeModelVars = TRUE,
  extraVarsToWrite = "ID"
)
```

Eine detaillierte Erläuterung der Funktion *rxPredict* finden Sie unter [23].

Aufgrund der Angabe von *writeModelVars = TRUE* enthält jeder Ausgabedatensatz die originalen Testdaten. Das Feld *ID*, das nicht Teil des Modells ist, wird wegen des Arguments *extraVarsToWrite = "ID"* ebenfalls zurückgegeben. Dieser Wert ist später in der Datenbank notwendig, um die Datensätze zuordnen zu können. Die vom Modell gemachten ▶



Bewertung des trainierten Modells. Dargestellt sind die vorausgesagten Wahrscheinlichkeiten für das Vorliegen eines bösartigen Tumors für bekanntermaßen gutartige (grün) und bösartige (rot) Tumoren (Bild 5)

Voraussagen werden in Form dreier Felder geliefert: *PredictedLabel* enthält die Klassifizierung (*benign* oder *malignant*), *Score.malignant* liefert eine Kennzahl (kleiner als 0 bedeutet *benign*, größer als 0 *malignant*).

Der Wert *Probability.malignant* stellt die Wahrscheinlichkeit dafür dar, dass eine Zuordnung zu *malignant auch* korrekt ist. In Bild 4 sehen Sie ein Beispiel für die Ausgabe von *predict_net*.

Damit Sie die Qualität des Modells einschätzen können, müssen Sie die Voraussage mit dem tatsächlichen Ergebnis vergleichen. Für einen schnellen Überblick geht das am besten grafisch mit der *plot*-Funktion. Tragen Sie die vorausge-

Listing 4: Biopsie-Tabelle anlegen

```
USE NeuralNetDemo
GO

CREATE TABLE Biopsy
(
  ID varchar(50) NOT NULL,
  V1 tinyint NULL,
  V2 tinyint NULL,
  V3 tinyint NULL,
  V4 tinyint NULL,

  V5 tinyint NULL,
  V6 tinyint NULL,
  V7 tinyint NULL,
  V8 tinyint NULL,
  V9 tinyint NULL,

  Class varchar(9) NULL,
  Class_predicted varchar(9) NULL,

  Score_malignant float NULL,
  Probability_malignant float NULL
  CONSTRAINT PK_BIOPSY PRIMARY KEY CLUSTERED (
    ID ASC)
)
GO
```

Listing 5: Tabelle für das trainierte Modell

```
# Datenbankverbindung definieren
conStr <- "
  Driver=SQL Server;
  Server=(local);
  Database=NeuralNetDemo;
  Trusted_Connection=true
"

# Objekt für Datenquelle erzeugen
ds <- RxSqlServerData(
  connectionString = conStr,
  table = "Models"
)

# Zunächst löschen, falls die Tabelle schon
# existiert
if (rxSqlServerTableExists("Models", conStr))
{
  rxSqlServerDropTable("Models", conStr)
}

# SQL definieren
ddl <- paste("
  CREATE TABLE Models
  (
    Name varchar(100) NOT NULL,
    Version varchar(10) NOT NULL,
    Model varbinary(max) NOT NULL)
")

ddl2 <- paste("
  ALTER TABLE Models
  ADD CONSTRAINT PK_Models
  PRIMARY KEY CLUSTERED (Name, Version)
")

# Tabelle anlegen
rxOpen(ds, "w")
rxExecuteSQLDDL(ds, ddl)
rxExecuteSQLDDL(ds, ddl2)

# Erst auf mode = read umstellen (Workaround, um
# Warnung zu umgehen)
rxOpen(ds, "r")
rxClose(ds)
```

sagten Werte für *Probability.malignant* gegen den Wertebereich 0 bis 1 auf, und zwar getrennt nach gutartigen und bösartigen Tumoren (Listing 3).

Bild 5 zeigt das Ergebnis. Im oberen Diagramm sind Voraussagen über jene Tumorproben dargestellt, die bekanntermaßen gutartig sind. Werte, die größer als 0,5 sind, werden vom Modell jedoch als bösartig eingestuft. Somit sind zwei falsch positive Diagnosen zu erkennen. Im unteren Diagramm sind Voraussagen über bösartige Tumorproben dargestellt. Hier liegt keine Fehlzuordnung vor – alle Werte liegen oberhalb einer Marke von 0,5. Dies ist insgesamt ein zufriedenstellendes Ergebnis.

	Name	Version	Model
1	rxNeuralNet	1.0	0x789CBDBB755CD44FF7284E7749374B0908D28D7477772E...

Das trainierte Modell wurde in der Datenbank abgelegt. Zu sehen ist ein Ausschnitt des Modells im Binärformat (Bild 6)

Bereitstellung und Anwendung

Bevor das neuronale Netz in die Datenbank überführt wird, legen Sie im SQL Server mithilfe des Skripts aus Listing 4 eine Tabelle an, in der neue Datensätze abgelegt werden kön- ▶

Listing 6: Speichern und Auslesen des trainierten Modells

```
# In die Tabelle schreiben
rxWriteObject(
  ds,
  "rxNeuralNet",
  trained_net,
  version = "1.0",
  keyName = "Name",
  valueName = "Model",
  versionName = "Version"
)

# Zur Kontrolle auslesen
rxReadObject(
  ds,
  "rxNeuralNet",
  version = "1.0",
  keyName = "Name",
  valueName = "Model",
  versionName = "Version"
)
```

Listing 7: Prozedur PredictBiopsy zum Aufruf des Modells

```
-- Zunächst löschen, falls die Prozedur schon existiert
DROP PROCEDURE IF EXISTS PredictBiopsy
GO

CREATE PROCEDURE PredictBiopsy
(
  @ModelName varchar(100),
  @Version varchar(10),
  @SampleData nvarchar(max)
)
AS
BEGIN
  DECLARE @serialisedModel varbinary(max) =
    (SELECT Model FROM Models WHERE Name =
      @ModelName AND Version = @Version)

  -- R-Skript ausführen
  EXEC sp_execute_external_script
    @language = N'R',
    @script = N'
      require("MicrosoftML")

      # Daten einlesen
      SampleData = InputDataSet

      # Modell laden (dekomprimieren und
      # deserialisieren)
      Model = rxReadObject(serialisedModel)

      # Modell einsetzen
      Prediction = rxPredict(Model, SampleData,
        extraVarsToWrite = "ID")
    ',
    @input_data_1 = @SampleData,
    @output_data_1_name = N'Prediction',
    @params = N'@serialisedModel varbinary(max)',
    @serialisedModel = @serialisedModel
  WITH RESULT SETS
  ((
    "ID" nvarchar(max),
    "Class_predicted" nvarchar(max),
    "Score_malignant" float,
    "Probability_malignant" float
  ))
END
GO
```

Listing 8: Funktionstest für PredictBiopsy

```
EXEC PredictBiopsy
  @ModelName = 'rxNeuralNet',
  @Version = '1.0',
  @SampleData =
  '
    SELECT * FROM
    (VALUES (''123'', 1, 1, 1, 1, 1, 1,
      1, 1, 1))
    AS Sample (ID, V1, V2, V3, V4, V5, V6,
      V7, V8, V9)
  '
```

	ID	Class_predicted	Score_malignant	Probability_malignant
1	123	benign	-3,1477746963501	0,0411791354417801

Ergebnis des Funktionstests aus Listing 8. Die Testdaten führten zur Diagnose (gutartig) (Bild 7)

nen. Die Tabelle spiegelt die Datenstruktur des Tumorbiopsie-Beispiels wider.

Anschließend öffnen Sie in RStudio eine Verbindung zum SQL Server und erstellen eine Tabelle namens *Models*, in der das trainierte Netzwerk abgelegt werden soll (Listing 5). Beim Schließen der Datenquelle kommt es bei der eingesetzten Version von R Services zu einer Warnung, die besagt, dass die Verbindung gar nicht geöffnet sei. Diese Warnung kann ignoriert werden und lässt sich auch umgehen, indem man den Öffnungsmodus zuvor auf *read* ändert.

Die Tabelle lässt sich selbstverständlich auch direkt im SQL Server anlegen; hier soll aber demonstriert werden, wie Sie SQL-Befehle von RStudio aus absetzen können.

Nach Erzeugen der Tabelle speichern Sie das Modell über die Funktion *rxWriteObject* in der Tabelle, wobei es serialisiert und komprimiert wird. Mit der *rxReadObject*-Funktion testen Sie, ob das Modell fehlerfrei abgerufen werden kann (Listing 6). Einen Blick in die *Models*-Tabelle der Datenbank offenbart das Modell im Binärformat, siehe Bild 6.

Legen Sie nun im SQL Server eine Stored Procedure namens *PredictBiopsy* gemäß Listing 7 an. Die Aufgabe dieser Prozedur ist es, Messwerte entgegenzunehmen, an ein beliebiges Modell weiterzureichen und die aus dem Modell stam-

Listing 9: Auswertung neuer Testdaten

```
-- Zunächst löschen, falls die Prozedur schon
-- existiert
DROP PROCEDURE IF EXISTS PredictNewBiopsies
GO

CREATE PROCEDURE PredictNewBiopsies
(
  @ModelName varchar(100),
  @Version varchar(10)
)
AS
BEGIN
  -- Nur ausführen, wenn neue Testdaten vorliegen
  IF EXISTS (SELECT * FROM Biopsy
    WHERE Class_predicted IS NULL)
  BEGIN
    -- Zwischentabelle für die Ergebnisse
    -- deklarieren
    DECLARE @result table
    (
      ID varchar(max),
      Class_predicted varchar(max),
      Score_malignant float,
      Probability_malignant float
    )

    -- Abfrage der Testdaten festlegen
    DECLARE @query varchar(max) =
    '
      SELECT ID, V1, V2, V3, V4, V5, V6,
        V7, V8, V9
      FROM Biopsy
      WHERE Class_predicted IS NULL
    '

    -- Aufruf der Modells und Füllen der
    -- Zwischentabelle
    INSERT INTO @result
      EXEC PredictBiopsy
        @ModelName = @ModelName,
        @Version = @Version,
        @SampleData = @query

    -- Aktualisierung der Originaltabelle
    UPDATE Biopsy
    SET Class_predicted =
      isnull(r.Class_predicted, 'NA'),
      Score_malignant = r.Score_malignant,
      Probability_malignant =
      r.Probability_malignant
    FROM @result r
    INNER JOIN Biopsy b ON b.ID = r.ID
  END
END
GO
```

menden Ergebnissätze zurückzuliefern. Dabei wird ein externes R-Skript aufgerufen, das innerhalb der Stored Procedure definiert ist [24].

Listing 8 zeigt, wie Sie testen können, ob die Prozedur erwartungsgemäß arbeitet. Sie übergeben ihr einfach einige Testdaten. Das Ergebnis sehen Sie in Bild 7.

Im Produktivbetrieb ist es wünschenswert, dass neue Messwerte lediglich in die Tabelle *Biopsy* eingetragen werden müssen. Ein zeitlich gesteuerter oder manuell angestoßener Job soll dann die Diagnosen hinzufügen. Hierzu legen Sie die Stored Procedure *PredictNewBiopsies* aus Listing 9 an. Sie ermittelt alle *Biopsy*-Datensätze, für die noch keine Diagnose gestellt wurde, übergibt die *ID* und die Messwerte *V1* bis *V9* an die Prozedur *PredictBiopsy*, speichert das Ergebnis in der Tabellenvariablen *@result* und aktualisiert zuletzt die Tabelle *Biopsy* mit den Ergebnissen. In Fällen, in denen das Modell keine Voraussage machen kann, weil ein Messwert fehlt, wird die Diagnose auf *NA* (nicht anwendbar) gesetzt.

Beim Aufruf von *PredictNewBiopsies* sind dann lediglich der Name und die Version des Modells anzugeben:

```
EXEC PredictNewBiopsies
  @ModelName = 'rxNeuralNet',
  @Version = '1.0'
```

Damit beschränkt sich die Auswertung neuer Tumorbiopsie-Daten auf das Ausführen einer gespeicherten Prozedur. Die Programmlogik des neuronalen Netzes liegt nahe bei den Daten und ist für außenstehende Komponenten transparent.

Fazit

Die Kombination aus Microsoft R Open, MicrosoftML, Net# und SQL Server funktioniert. Es ist erstaunlich einfach, eine Datenbank mit maßgeschneiderten künstlichen neuronalen Netzen auszustatten und damit datenbankbasierte Anwendungen um lernfähige Algorithmen zu erweitern. Wer SQL Server ohnehin einsetzt, der benötigt in Zukunft auch keine weitere Serversoftware. Damit wird die Einstiegshürde zur Entwicklung „smarter" Anwendungen im Microsoft-Umfeld deutlich gesenkt.

Was sicherlich noch fehlt, ist eine Integration in die .NET-Welt. Derzeit führt SQL Server (beziehungsweise die R-Laufzeitumgebung) lediglich R-Skripte aus – man muss sich also nicht nur mit der Technologie R vertraut machen, sondern hat mitunter mit den üblichen Eigenschaften zu kämpfen, die Skript-Technologien mit sich bringen (zum Beispiel dynamische Typisierung und eingeschränktes Debugging). Auch müssen die Skripte in Stored Procedures eingebunden werden, was kein besonders eleganter Ansatz ist.

Aber Microsoft R steht erst am Anfang. Interessant wird das Einbinden von R in die Entwicklungsumgebung mittels der R Tools for Visual Studio. ■

[1] Mykola Dobrochynskyy, Künstliche Intelligenz, Eine neue Ära, dotnetpro 3/2017, Seite 8 ff., www.dotnetpro.de/A1703KI

[2] Wie Maschinen lernen lernen, www.dotnetpro.de/SL1710MicrosoftML1

[3] Künstliches neuronales Netz, Anwendung, www.dotnetpro.de/SL1710MicrosoftML2

[4] Alpha Go zerlegt Go-Community, www.dotnetpro.de/SL1710MicrosoftML3

[5] Netzgespinste, www.dotnetpro.de/SL1710MicrosoftML4

[6] Überblick Ansätze des Deep Learning, www.dotnetpro.de/SL1710MicrosoftML5

[7] Introduction to MicrosoftML, www.dotnetpro.de/SL1710MicrosoftML6

[8] The R Project for Statistical Computing, www.r-project.org

[9] Microsoft R Application Network, http://mran.microsoft.com

[10] Introducing Microsoft R Client, www.dotnetpro.de/SL1710MicrosoftML7

[11] Introducing Microsoft R Server, www.dotnetpro.de/SL1710MicrosoftML8

[12] SQL Server R Services, www.dotnetpro.de/SL1710MicrosoftML9

[13] MASS: Support Functions and Datasets for Venables and Ripley's MASS, www.dotnetpro.de/SL1710MicrosoftML10

[14] RStudio, www.rstudio.com

[15] R Tools for Visual Studio, http://microsoft.github.io/RTVS-docs

[16] SQL Server Evaluations, www.dotnetpro.de/SL1710MicrosoftML11

[17] SQL Server Management Studio (SSMS) – Release Candidate, www.dotnetpro.de/SL1710MicrosoftML12

[18] RClient Download, http://aka.ms/rclient/download

[19] Download RStudio, www.dotnetpro.de/SL1710MicrosoftML13

[20] Guide to Net# neural network specification language for Azure Machine Learning, www.dotnetpro.de/SL1710MicrosoftML14

[21] rxNeuralNet: Neural Net, www.dotnetpro.de/SL1710MicrosoftML15

[22] Getting started with GPU acceleration for MicrosoftML's rxNeuralNet, www.dotnetpro.de/SL1710MicrosoftML16

[23] rxPredict.mlModel: Score using a Microsoft R Machine Learning model, www.dotnetpro.de/SL1710MicrosoftML17

[24] Verwenden von R-Code in Transact-SQL (SQL Server R Services), www.dotnetpro.de/SL1710MicrosoftML18

Martin Gossen
ist IT-Berater bei der IKS GmbH in Hilden. Er erstellt seit 15 Jahren Softwarelösungen auf Basis von C#, .NET und Microsoft SQL Server. Sie erreichen Ihn unter **m.gossen@iks-gmbh.com**.

dnpCode A1710MicrosoftML