Machine Learning mit H2O und Java

Stephan Schiffner und Dr. Jonathan Boidol, Steadforce

Schätzungsweise 60 Prozent aller Machine-Learning- und Analytics-Projekte in Unternehmen schaffen es nicht über Experiment- oder Pilotphasen hinaus [1]. Eine der Herausforderungen ist die Distanz zwischen den vertrauten Entwicklungsumgebungen von Statistikern, Analytics-Experten und Data Scientists, hinzu kommt der üblichen Technologie-Stack von Software-Entwicklern, die für Umsetzung und Einsatz in Produktivsystemen verantwortlich sind. Eine mögliche Lösung dafür ist H2O, eine Machine-Learning-Plattform, die es ermöglicht, beide Welten einfach miteinander zu verbinden. Dieser Artikel stellt beispielhaft vor, wie mit H2O ein Machine-Learning-Modell in der Statistiker-Sprache R entwickelt und in Java auf Streaming-Daten live angewendet wird. Zum Einsatz kommen H2O in R, Spark Streaming in Java und Livedaten aus einem Twitter-API.

Der prototypische Workflow eines Data Scientist sieht etwa folgendermaßen aus: Er bekommt einen Datensatz, der manuell annotiert, geprüft, gelabelt, kuratiert oder auf andere Weise angereichert ist, füttert diese Daten in einen geeigneten Algorithmus und erhält daraus ein fertig trainiertes Modell. Dieses Modell kann jetzt auf Daten von der gleichen Art angewandt werden, die allerdings noch nicht manuell annotiert wurden. Ziel ist es, die Arbeit des Menschen zu replizieren, mit weniger Aufwand, auf größeren Datenmengen und im besten Fall sogar mit größerer Genauigkeit und Zuverlässigkeit.

Ein konkretes Beispiel wäre das Unterscheiden von Objekten auf Bildern: Wir haben einen Datensatz von ein paar Hundert Bilder, auf denen entweder eine Rose oder eine Tulpe zu sehen ist, jedes Bild mit einem passenden Label versehen. Das Modell kann nach dem Training auf neuen Bildern ohne Label entscheiden, ob diese eine Rose oder eine Tulpe zeigen. Solche Klassifikationsaufgaben sind typische Anwendungsgebiete und ein wichtiger Teilbereich von Machine Learning *(siehe Abbildung 1)*.

Wer solche Modelle entwickelt, hat oft einen Hintergrund als Mathematiker, Statistiker oder Physiker und arbeitet mit Werkzeugen,

Label	Tweet-Inhalt
Negativ	"Uqhhh it's sooooo hot outside ! #Texasweather"
Neutral/teilt Informationen	"Weather Alert: Flood Warning issued May 22 at 6:32PM MDT expiring May 23 at 9:32AM MDT by NWS Glasgow {link}... {link}"
Positiv	"Friday evening. Weather? Beautiful. Last man standing at the office. Bitter? Nah. It only makes payday that much sweeter. Stay hungry."
Nicht wetterbezogen	"Community Blood Center Media Blood drive Tues Noon-6 at Westridge Mall by Penneys lower level - Blood to help Joplin st ..."

Tabelle 1: Beispiele für Tweets mit Label

die für solche Modellierungsaufgaben und Datenexploration ausgerichtet sind. Häufig zum Einsatz kommen etwa Python oder das unter Statistikern weitverbreitete R *[2]*.

Im Gegensatz dazu arbeiten Software-Entwickler in Sprachen, Technologien und Umgebungen, die auf ihre spezifischen Anforderungen zugeschnitten sind. Wie ist es möglich, die Analyse-Ergebnisse oder Machine-Learning-Modelle, entstanden etwa in R, sinnvoll in ein Produktivsystem, realisiert etwa als Java-Backend, zu übernehmen? H2O erlaubt es uns, in R entwickelte Modelle in die Java-Welt zu bringen, ohne sie mühsam und fehleranfällig nachzubauen. Das wird an einem Beispiel zur Analyse von Textnachrichten gezeigt.

Demo Use Case: Tweet Klassifikation

Eine der schwierigeren, aber auch interessanteren Aufgaben aus dem Bereich „Machine Learning und Data Mining" ist es, menschliche Stimmungen und Gefühle zu erkennen – man spricht dabei auch von Sentiment-Analyse. Mit den entsprechenden Trainingsdaten und Werkzeugen werden wir einen Versuch in diese Richtung machen. Wir sehen uns Mitteilungen an, die Nutzer bei Twitter veröffentlichen, um zu erkennen, welche Einstellung jemand über das aktuelle Wetter hat – ist es ihr oder ihm zu heiß, zu kalt oder zu nass oder freuen sich die Twitterer im Gegenteil gerade über den herrlichen Sonnenschein. Dazu verwenden wir einen freien Satz von Trainingsdaten, in dem tausend Tweets per Hand in eine von vier Klassen eingeteilt wurden, nämlich positive, negative, neutrale Einstellungen zu Wetter sowie zur Abgrenzung auch einige Beispiele von Tweets, die nichts mit dem Wetter zu tun haben. Der Datensatz ist unter *[3]* verfügbar.

Tabelle 1 zeigt ein paar Beispiele solcher gelabelter Tweets. Ziel ist es, neue, noch nicht gelabelte Tweets einer dieser vier Klassen zuzuordnen. Dazu verwenden wir die Machine-Learning-Werkzeuge, die uns H2O zur Verfügung stellt. Um diese Sentiment-Vorhersage live durchzuführen, nutzen wir Spark Streaming, um in dieser Umgebung laufend die neuesten Tweets zu klassifizieren.

H2O ist eine vergleichsweise neue Plattform für Machine Learning, die bereits einige sehr fortschrittliche Fähigkeiten bietet und kürzlich auch zum Leader im Gartner-Magic-Quadrant für Data-Science- und Machine-Learning-Plattformen aufgestiegen ist. Die Open-Source-Lösung wird von einigen bekannten Stanford-Professoren beraten, Freunde des statistischen Lernens kennen sicher Namen wie „Tibshirani" und „Hastie".

H2O bietet nicht nur die üblichen Standard-Algorithmen und einige „State-of-the-art"-Methoden an, bei der Implementierung wurde auch sehr auf Effizienz und Performance geachtet, etwa durch interne Komprimierung von Daten. H2O arbeitet In-Memory und skaliert als ML-Plattform auf verteilte Rechencluster.

Eines der interessanten Features sind die Schnittstellen: H2O lässt sich in verschiedenen Programmiersprachen einsetzen, vorneweg Java, Python, R und Scala, genauso lässt es sich in weitere Tools wie Tableau und Spotfire einbinden. Das funktioniert, weil im Hintergrund ein (in Java implementierter) H2O-Cluster läuft. Dieser kann lokal aufgebaut werden oder auf mehrere Nodes verteilt sein. Die Funktionsaufrufe, mit denen ein Programm H2O verwendet, werden in REST-Calls an den Cluster übersetzt, die jeweiligen Program-

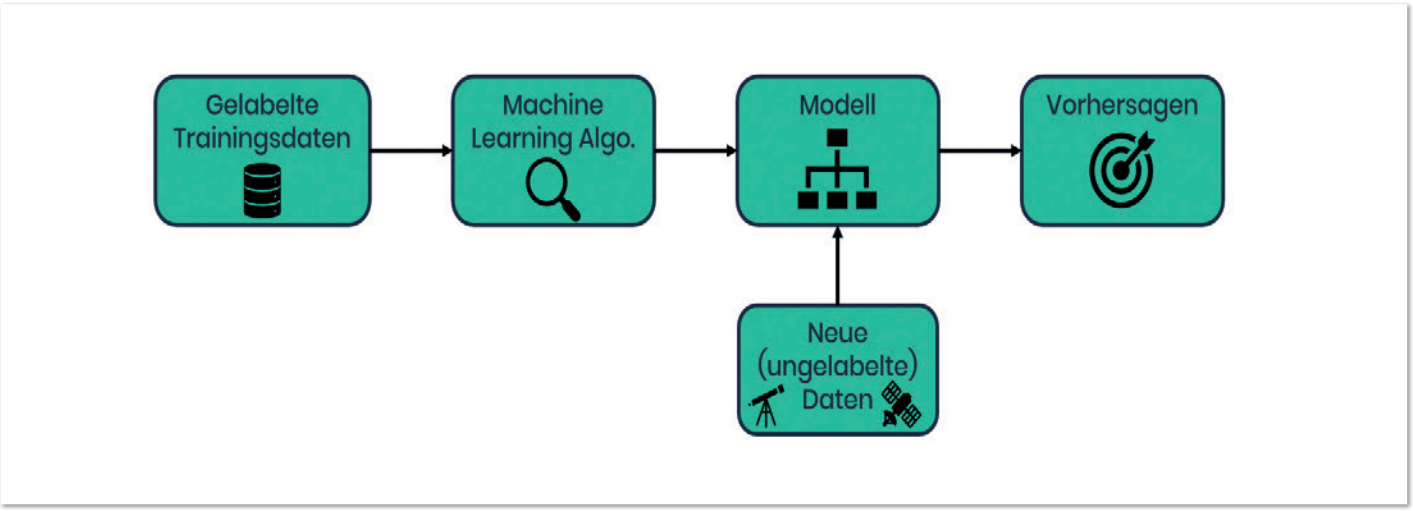

Abbildung 1: Schema für das Lernen und Anwenden eines Klassifikators

```
library(h2o)
h2o_context <- h2o.init(ip="localhost", port=4321)
```

Listing 1

miersprachen dienen als Interfaces und Operationen werden etwas versteckt innerhalb des Clusters ausgeführt. Das ermöglicht einen hohen Grad von Abstraktion und große Interoperabilität verschiedener Umgebungen über H2O *(siehe Abbildung 2)*.

Modell-Entwicklung in R

Wir starten unsere Pipeline zunächst weit weg von Java und verwenden R, um die Daten vorzubereiten und unser Modell zu trainieren. Als Entwicklungsumgebung bietet sich hier das weitverbreitete R-Studio an, es gibt allerdings auch Plug-ins für Eclipse, die R-Unterstützung ermöglichen.

Zunächst binden wir die H2O-Library ein und verbinden uns mit dem „init()"-Befehl zu einem H2O-Cluster. Läuft an der angegebenen Adresse noch kein Cluster, wird direkt eine lokale Instanz erzeugt *(siehe Listing 1)*.

Der erste Arbeitsschritt besteht darin, die Daten vom Filesystem in den Cluster zu übertragen. Das geschieht für uns sehr einfach mit „importFile()", und zwar unabhängig von der Datenquelle, auch wenn die Daten etwa aus einem Hadoop-System kommen. Im Hintergrund wird in beiden Fällen nur ein REST-Aufruf an den H2O-Cluster getätigt, die Details wurden für uns wegabstrahiert. Im Ergebnis wird ein H2O-Dataframe im Speicher des Clusters abgelegt. Dabei handelt es sich um ein spezielles, Tabellen-ähnliches Datenformat von H2O, das stark dem Dataframe ähnelt, der in R die Standard-Struktur für Datensätze aller Art ist *(siehe Listing 2)*.

Anstatt direkt auf den Tweets in Textform zu arbeiten, führen wir sie in ein Word-Embedding über. Jeder Text lässt sich mit einem solchen Embedding in einem Vektorraum darstellen, in dem ähnliche Texte auf nahe zusammenliegende Wörter abgebildet werden. Der Vorteil dabei ist einerseits, dass wir Texte mit einem Vokabular von vielleicht 100.000 oder mehr Wörtern kompakt darstellen können, hier im Beispiel in einem 100-dimensionalen Vektor. Andererseits erleichtert diese Transformation auch die Klassifikation der Tweets, da eben ähnliche Texte in ähnliche Bereiche des Vektorraums abgebildet werden, also gewissermaßen schon vorsortiert sind. Auch um Synonyme, Verneinungen und Wortkombinationen müssen wir uns nicht separat kümmern.

Ganz praktisch müssen wir die Wörter der Tweets zunächst etwas bereinigen, trennen sie also erst an Leerzeichen, entfernen zu kurze Wörter und Zahlen und vereinheitlichen zu Kleinbuchstaben. Die Hilfsfunktion dazu definieren wir in R mit den Tools von H2O. Das Training des Word-Embedding erfolgt wieder mit den High-Level-Funktionen von H2O, in unserem Fall haben wir als Modell Word2Vec mit der passend benannten Funktion „word2vec()" gewählt, ein Embedding auf Basis eines neuronalen Netzes *(siehe Listing 3)*.

```
# aus Filesystem:
twitter_data_df <- h2o.importFile("weather-agg-DFE.csv", header = T)
# aus Hadoop:
twitter_data_hdfs <- "hdfs://node-1:/user/data/twitter/weather-agg-DFE.csv"
twitter_data_df <- h2o.importFile(twitter_data_hdfs, header = T)
```

Listing 2

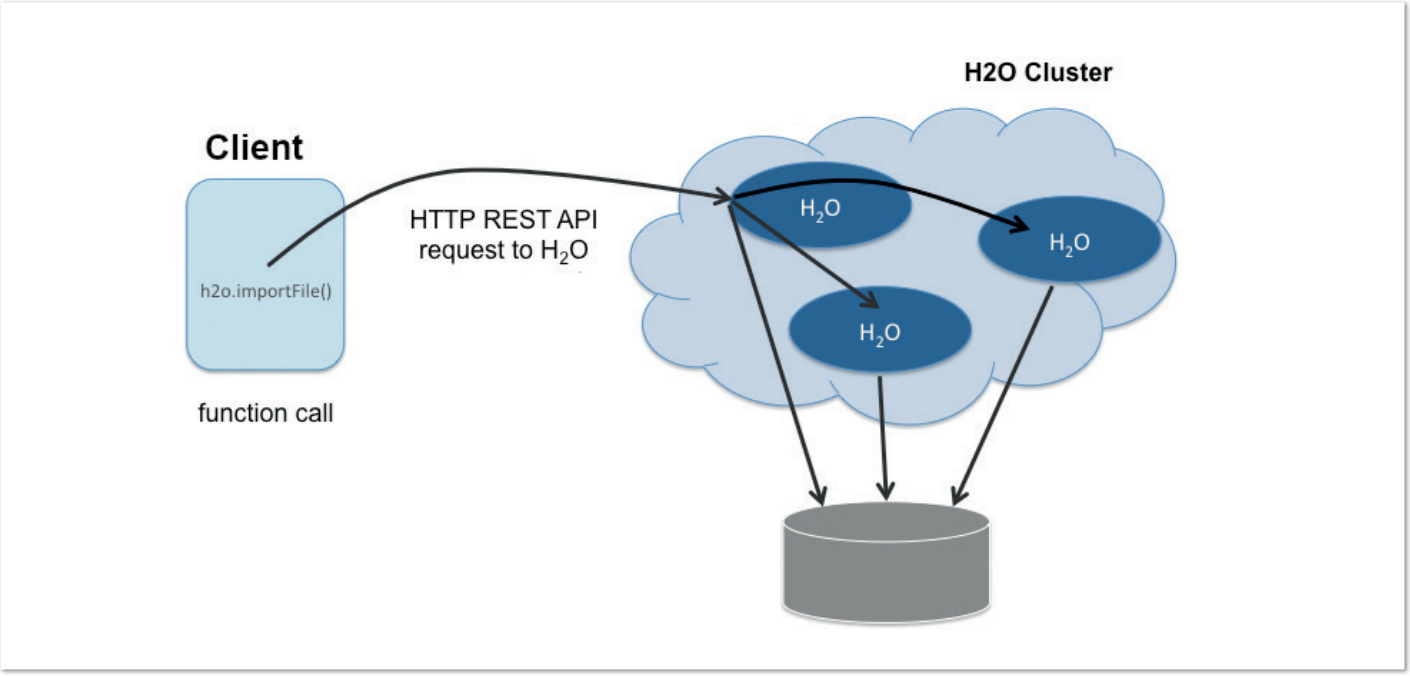

Abbildung 2: H2O-Cluster-Kommunikation [4]

```
tokenize <- function(sentences) {
  tokenized <- h2o.tokenize(sentences, "\\\\W+")
  tokenized.lower <- h2o.tolower(tokenized)
  tokenized.lengths <- h2o.nchar(tokenized.lower)
  tokenized.filtered <- tokenized.lower[is.na(tokenized.lengths) || tokenized.lengths >= 2,]
  tokenized.words <- tokenized.filtered[h2o.grep("[0-9]", tokenized.filtered, invert = TRUE, output.logical =
TRUE),]
  tokenized.words[is.na(tokenized.words),]
}
tweets_tokenized <- tokenize(twitter_data_df$tweet_text)

w2v_model <- h2o.word2vec(tweets_tokenized, vec_size = 100)
tweets_vecs <- h2o.transform(w2v_model, tweets_tokenized)
twitter_data_embedded <- h2o.cbind(twitter_data_df$emotion, tweets_vecs)
```

Listing 3

Wir haben nun einen fertig aufbereiteten Datensatz bestehend aus tausend Tweets, dargestellt als numerische Vektoren, und dem jeweils zugehörigen Label, das uns sagt, welche Emotion durch den Tweet ausgedrückt wurde. Diesen Datensatz können wir zum Training eines Klassifikationsmodells einsetzen. Wir verwenden ein Modell auf Basis von Entscheidungsbäumen. Etwas vereinfacht bauen wir dabei eine hierarchische Baumstruktur, in der wir für jeden neuen Tweet einem Pfad folgen können, der uns zu einem Label führt, das wir als Vorhersage benutzen wollen *(siehe Abbildung 3)*.

Ein kleiner Baum auf Basis von Stichwörtern, die im Tweet entweder vorkommen oder fehlen, könnte wie in *Abbildung 2* aussehen: Erscheint das Stichwort „sun" im Tweet, folgen wir der rechten Abzweigung. Jetzt könnte dem Autor aber auch zu heiß sein, erkennbar am Stichwort „too". Abhängig davon entscheiden wir uns für ein positives oder ein negatives Label. In unserem Beispiel verwenden wir einen Algorithmus, der viele solcher Bäume parallel erzeugt und ihre Vorhersagen kombiniert. Um dieses, „Random Forest" genannte, Modell zu trainieren, geben wir die vorherzusagende Variable an sowie den Datensatz, von dem gelernt werden kann. Das sind eben die Emotion des Tweets und die Tweet-Vektoren. Diese Angaben sind für alle Arten von Modellen nötig. Speziell für den Random Forest spezifizieren wir noch einige Parameter über die Zahl und Art der zu trainierenden Bäume *(siehe Listing 4)*.

An dieser Stelle könnten wir noch weitere Schritte zur Modell-Entwicklung durchführen. Wir können versuchen, die Parameter des Modells optimal einzustellen oder andere Modelle ausprobieren. Ein sehr wichtiger Punkt ist auch die Validierung des Modells. Dabei wird geprüft, ob das trainierte Modell auch auf neue Daten sinnvoll anwendbar ist, also nicht nur die Trainingsdaten auswendig gelernt hat, sondern gut generalisiert. Für unser Beispiel soll uns das jedoch nicht weiter interessieren. Die wichtigere Frage lautet, wie wir das fertige Modell weiterverwenden können.

Unser Ziel ist es, das Modell auf Live-Daten in Spark anzuwenden. Dazu könnten wir die Spark-Integration von H2O verwenden, die als „Sparkling Water" verfügbar ist. Aber eigentlich wollen wir die Welt von R verlassen und auch der H2O-Cluster ist nicht zwingend nötig, um die Modelle weiterzuverwenden. Stattdessen exportieren wir das fertige Modell in Java-kompatiblem Format. Dazu bietet H2O grundsätzlich zwei Möglichkeiten an: POJOs und MOJOs. *Tabelle 2*

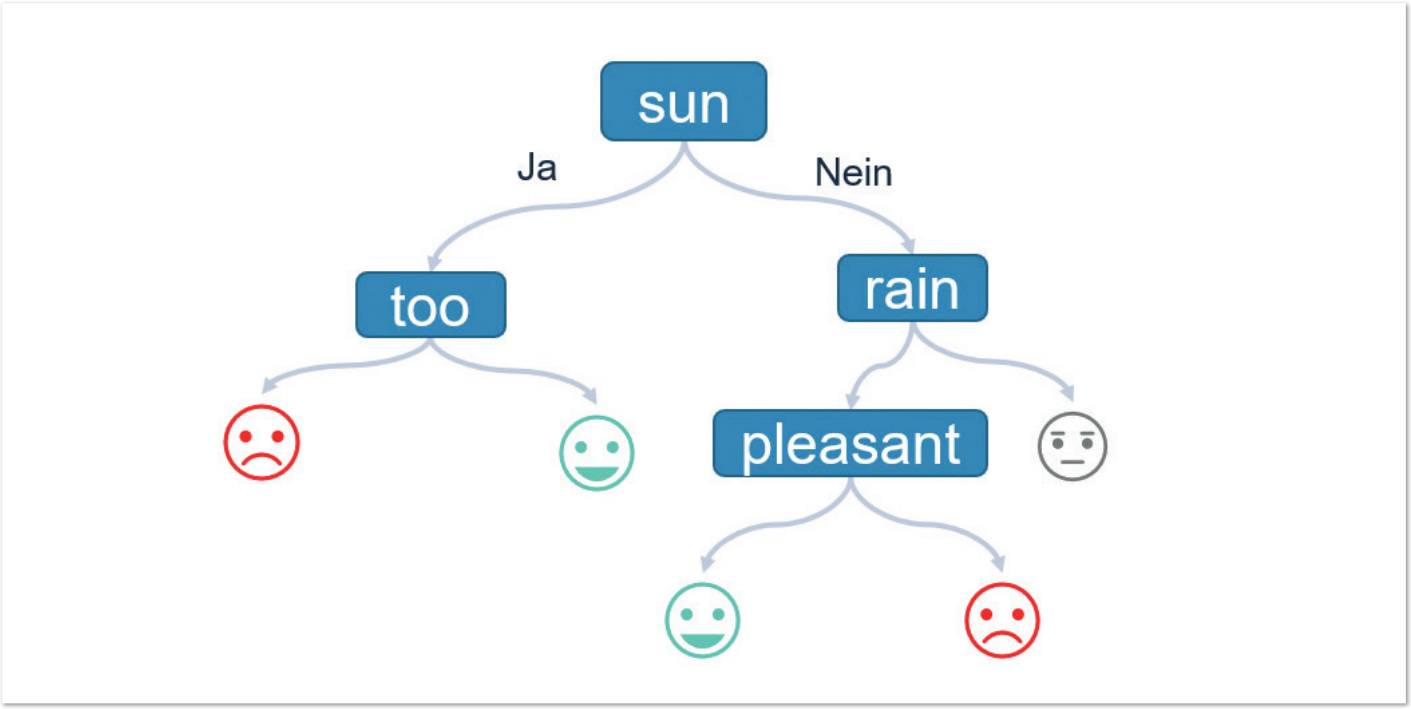

Abbildung 3: Einfacher Entscheidungsbaum

```
target <- "emotion"
emotion_model <- h2o. randomForest(y = target,
                        training_frame = twitter_data_embedded,
                ntrees = 100,
                        max_depth = 15)
```

Listing 4

Format	Typ	Eigenschaften
POJO	- Plain Old Java Object - kompilierbare Java-Klassen, die das Modell abbilden	- Java-Versions-unabhängig - Quellcode inspizierbar - Hängt von jar-file mit H2O-Klassen ab
MOJO	- Model Object Optimized - Binär-Format	- Platzsparend - Erst für wenige Modelle verfügbar - Hängt von jar-file mit H2O-Klassen ab

Tabelle 2: Modell-Exportformate von H2O

gibt einen kurzen Überblick über diese beiden Formate. Beide lassen sich direkt in Java als native Objekte verwenden, benötigen allerdings zusätzlich einen Satz von H2O-Java-Dependencies. Diese lassen sich mit unserem letzten Call zum H2O-Cluster zusammen mit dem MOJO oder POJO erzeugen.

Unser Modell wird dann als MOJO-File abgelegt, das „dependencies.jar" enthält die nötigen Klassen, um es in Java anzuwenden. Genauso exportieren wir auch das Word2Vec-Modell, das die Tweets in Vektoren transformiert *(siehe Listing 5)*.

Live-Klassifikation mit Spark Streaming

Wir wollen unser Modell zur Klassifikation von neuen Tweets einsetzen. Diese werden kontinuierlich erzeugt und stehen über ein kostenloses API von Twitter als Datenstrom zur Verfügung. Diesen Datenstrom bewältigen wir mit Spark Streaming, für unser Beispiel wieder möglichst einfach gehalten. Auf ein paar Konfigurationsschritte und Implementierungsdetails mussten wir aus Platzgründen verzichten, der Quellcode im Folgenden zeigt jedoch alle wichtigen Schritte.

Apache Spark ist eine der populärsten Plattformen für Cluster-Computing, das zusätzliche Streaming-Modul erlaubt die effiziente Verarbeitung von eben nicht nur Batch-, sondern Streaming-Daten. Die zugrunde liegende Spark-Architektur erlaubt es, die anfallende Last auf Server-Nodes zu verteilen. Einer der Vorteile von Spark und damit auch von Spark-Streaming ist die Verarbeitung In-Memory, was hohen Datendurchsatz ermöglicht. Die grundlegende Datenstruktur in Spark Streaming sind DStreams, eine verteilte Datensammlung, die man sich wie eine Collection von Records (streng typisiert) vorstellen kann.

Das Setup und die Programmierung von Spark Streaming erfolgen wieder vollständig in Java: Wir erzeugen einen Spark-Streaming-Context, der als Einstiegspunkt für die gesamte Spark Engine dient. Diesem geben wir eine Batchdauer mit, im Beispiel zwei Sekunden. Das bedeutet, dass Spark Streaming für zwei Sekunden Daten sammelt und diese als Mini-Batches weiterverarbeitet, was wieder die Performance gegenüber anderen Streaming Engines deutlich erhöht.

Um die echten Twitter-Daten in Spark Streaming einzubinden, verwenden wir die TwitterUtils-Bibliothek *[5]*. Um nicht von einem Tweet-Sturm überwältigt zu werden, filtern wir an dieser Stelle schon Tweets heraus, die ein paar Wetter-bezogene Keywords enthalten. „createStream()" erzeugt dann einen DStream, der laufend die aktuellen Tweets des Mini-Batch enthält *(siehe Listing 6)*.

Als nächsten Schritt binden wir die Modelle ein, die wir aus dem H2O-Cluster exportiert haben. Da es sich letztlich um Java-Objekte handelt, ist das wieder ganz einfach und geschieht mit den Hilfsfunktionen, die wir in den H2O-Dependencies finden *(siehe Listing 7)*.

```
modelfile <- h2o.download_mojo(emotion_model,
                               path = "/model/dir",
                               get_genmodel_jar = T,
                               genmodel_n = "dependencies.jar")
```

Listing 5

```
SparkConf sparkConf = new SparkConf().setMaster("local[4]").setAppName("StreamClassificationDemo");
JavaStreamingContext ssc = new JavaStreamingContext(sparkConf, Durations.seconds(2));
String[] filters = { "weather", "rain", "sunshine" };
JavaReceiverInputDStream<Status> tweetStream = TwitterUtils.createStream(ssc, filters);
```

Listing 6

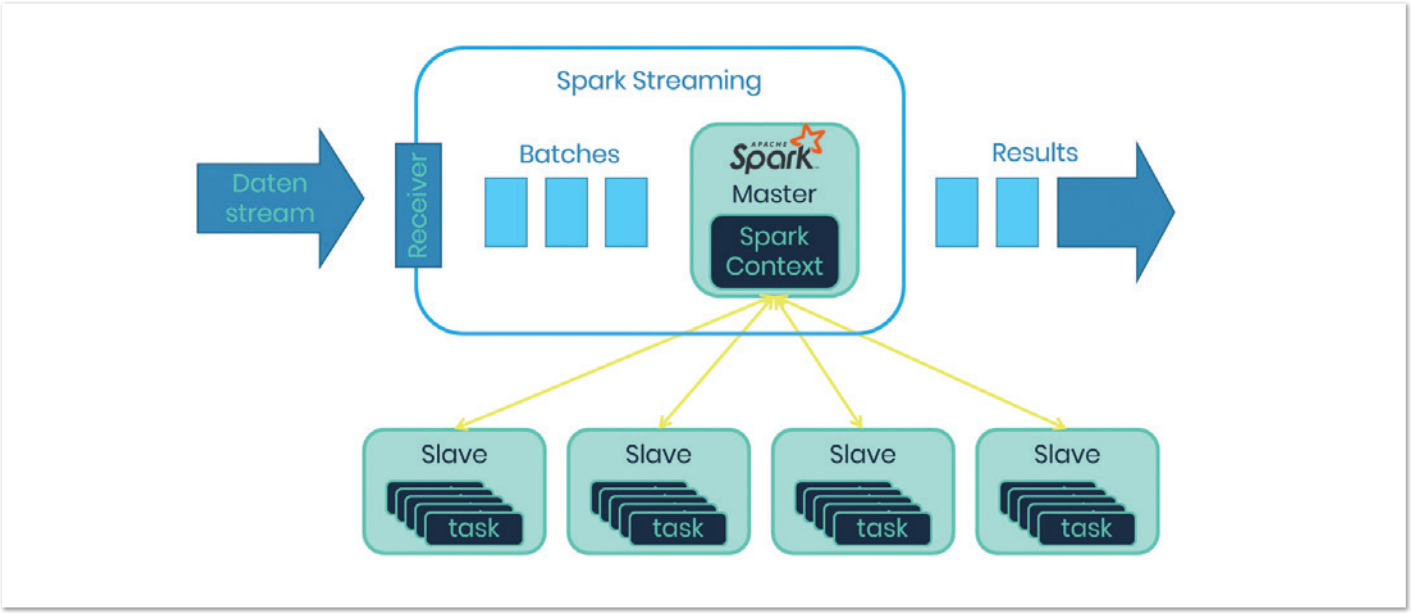

Abbildung 4: Spark-Streaming-Architektur

```
EasyPredictModelWrapper predictionModel = new EasyPredictModelWrapper(
  MojoModel.load("static/emotion_model.zip"));
WordEmbeddingModel w2vModel = Word2VecMojoModel.load("static/Word2Vec_model.zip");
```

Listing 7

Für das eigentliche Scoring, also die Klassifikation, der Tweets schreiben wir wieder eine Java-Hilfsfunktion, um die Tweets, schon in einzelne Wörter tokenisiert als Liste von Strings, zunächst wieder mithilfe des Word2Vec-Modells in einen numerischen Vektor zu übersetzen und diesen Vektor dann an das trainierte Klassifikations-Modell weiterzureichen *(siehe Listing 8)*.

Auf den Spark-Streaming-DStream angewandt, wird die Klassifikation dann für jeden Tweet, der über das API hereinkommt, ausgeführt. Das Modell muss auf jedem der Nodes vorhanden sein *(siehe Listing 9)*.

Wir haben Spark Streaming konfiguriert, eine Datenquelle angebunden, die Prädiktionsmodelle eingebunden und wenden diese auf die Streaming-Daten an – damit ist unsere Pipeline fertig definiert. Zu guter Letzt können wir die gesamte Pipeline endlich anwerfen und die Resultate beobachten. Dazu starten wir den Spark-Streaming-Context und lassen so lange Batches der Twitterdaten verarbeiten,

```
private static void classifyTweet(ArrayList<String> tweet, EasyPredictModelWrapper predictionModel,
        WordEmbeddingModel w2vModel) throws PredictException {
     final int vecSize = w2vModel.getVecSize();
     float[] tweetEmbedding = new float[vecSize];
     tweet.stream().map(word -> w2vModel.transform0(word, new float[vecSize])).reduce(tweetEmbedding,
           Utils::sumFloatarrays);

     RowData row = new RowData();
     for (int i = 0; i < vecSize; i++) {
        row.put("C" + i+1, Float.toString(tweetEmbedding[i]));
     }
     MultinomialModelPrediction prediction = predictionModel.predictMultinomial(row);
     System.out.println(prediction.label + ":");
}
```

Listing 8

```
tweetStreamTokenized.foreachRDD(rdd -> {
        rdd.foreachPartition(tweet -> {
           while (tweet.hasNext()) {
              classifyTweet(tweet.next(), predictionModel, w2vModel);
           }
        });
     });
```

Listing 9

Vorhersage	Tweet-Inhalt
Negativ	▪ "daijah gay ass pulled me out of work to kiss under the fucking rain" ▪ "england winning yday isnt enough to justify the weather"
Neutral/teilt Informationen	▪ "enjoying this glorious weather in the uk fond memories of this time last week" ▪ "Wind 0 mph gusting to 0 mph. Temperature 71.5 F. Humidity 84%. Rain Today: 0.00in"
Positiv	▪ „happy birthday america wishing you all fair weather safe shenanigans and quality time with friends and family" ▪ "I can't complain with 6 weeks off and lots of sunshine. Hope you're enjoying the summer break."
Nicht wetterbezogen	▪ "allen is going to hell for the last one" ▪ "is your caption about the weather or yourself?"

Tabelle 3: Einige Beispiele für klassifizierte Tweets

```
ssc.start();
ssc.awaitTermination();
```

Listing 10

bis wir das Programm abbrechen *(siehe Listing 10)*. Die *Tabelle 3* zeigt ein paar unserer Vorhersagen – nicht alle sind perfekt, einige der Tweets auch mehrdeutig, aber insgesamt ist das Ergebnis recht zufriedenstellend.

Natürlich bietet H2O noch viel mehr Funktionalitäten für Machine Learning, als hier gezeigt sind. Wer sich für neuronale Netze interessiert, findet hier etwa die Möglichkeit, auch solche Modelle zu entwickeln. Die angesprochene Parameter-Optimierung von Modellen ist mit mehreren Strategien wie Grid-Search oder Random-Discrete-Search möglich. Wer sich nicht für ein Modell entscheiden möchte, kann in sogenannten „Ensembles" einfach mehrere Modelle optimal miteinander kombinieren. Dazu kommen natürlich die üblichen Toolkits zu Modell-Validierung, Daten-Vorverarbeitung und -Cleaning. Wem das nicht genügt, der kann immer auf die nativen Möglichkeiten der gewählten Schnittstellensprache zurückgreifen.

Take aways

In der Praxis laufen Modellentwicklung und Produktivanwendung in unterschiedlichen Geschwindigkeiten, Arbeitsumgebungen und mit anderen Methoden und Zielen. Letztlich existieren also zwei parallele, getrennte Pipelines, wie wir auch in unserem Klassifikationsbeispiel gesehen haben. Den Output der ersten Pipeline, nämlich das Ergebnis von Analysen und fertige Modelle, in die zweite Pipeline zu bringen, ist keine triviale Aufgabe. Einzelne Elemente sind nicht übertragbar und müssen im schlimmsten Fall neu- oder nachimplementiert werden. Die H2O-Plattform bietet hier eine einfache Lösung, indem sie den Export und Transfer in die Java-Welt unterstützt oder selbst als Analytics-Backend einsetzbar ist. Dank der Unterstützung von Cloud Services wie AWS und Cluster-Computing wie Spark lassen sich auch Projekte mit großen Datenmengen mit H2O umsetzen.

Weitere Informationen

[1] *https://www.gartner.com/newsroom/id/3130017*
[2] *https://www.r-project.org*
[3] *https://data.world/crowdflower/weather-sentiment*
[4] *https://www.h2o.ai/h2o*
[5] *http://bahir.apache.org/docs/spark/current/spark-streaming-twitter*

Stephan Schiffner
stephan.schiffner@steadforce.com

Stephan Schiffner ist Director Advanced Analytics bei Steadforce und gesamtverantwortlich für Kunden-Projekte sowie die strategische Weiterentwicklung des Bereichs. Neben seinen beruflichen Aufgaben ist er seit dem Jahr 2011 als Lehrbeauftragter an der Hochschule München in den Bereichen „Software-Entwicklung" und „Software Engineering" aktiv.

Dr. Jonathan Boidol
jonathan.boidol@steadforce.com

Jonathan Boidol hat einen Master of Science in Bioinformatik erworben und an der LMU München zur Analyse von Streaming-Daten promoviert. Er ist Autor internationaler Fachartikel über Data Mining und Machine Learning. Jonathan Boidol ist Data Scientist bei Steadforce und arbeitet in Projekten von der Use-Case-Evaluierung bis zum Design und zur Implementierung von Advanced-Analytics-Modellen.