

Fortgeschrittene Datenbanktechniken

Big Data - Hadoop - NewYorkCity Taxi Trips



Geschrieben von:

Dennis Ahrens

Fabian Kalkofen

Philipp Kamps

Dominic Prass

Johannes Schapdick

David Schlebes

Michael Nienhaus

Inhalt


1	Einleitung & Motivation	1
1.1	Big Data	2
2	Aufgabenstellung	3
2.1	Analyse	3
2.2	Prognose.....	4
3	Technologien	5
3.1	Hadoop	5
3.1.1	HDFS	5
3.1.2	MapReduce	6
3.1.3	Hive.....	8
3.2	Amazon Web Services (AWS)	9
3.3	R Programmiersprache.....	10
4	Daten	10
5	Implementierung.....	13
5.1	Analyse	13
5.2	Prognose.....	18
5.2.1	Bereichsdefinition	19
5.2.2	Selektion der relevanten Daten	20
5.2.3	Auswahl des Data-Mining-Verfahrens	22
5.2.4	Prognoseerstellung – Erster Ansatz	24
5.2.5	Prognoseerstellung – Zweiter Ansatz.....	26
5.2.6	Erkenntnis.....	28
6	Fazit	30
7	Ausblick	32
8	Anhang	33



1 Einleitung & Motivation

In der Veranstaltung Fortgeschrittene Datenbank-Techniken wird das Thema Big-Data genauer beleuchtet. Heutzutage ist Big-Data im Umfeld von Unternehmen nicht mehr wegzudenken und auch politisch von großem Interesse. Große Firmen nutzen die Informationen von ihren Nutzern um zum Beispiel Werbung zu verbessern oder Auswirkungen des Umfelds auf Ihre Verkaufszahlen zu analysieren. Der Staat betreibt beispielsweise durch die Vorratsdatenspeicherung Verbrechensbekämpfung, Verbrechensaufklärung oder Terrorismusabwehr.

Da das Datenaufkommen im Allgemeinen stetig zunimmt, stoßen herkömmliche relationale Datenbanksysteme an ihre Grenzen. Sie können Datenmengen in solcher Größe nicht mehr performant verarbeiten. Beispielsweise verarbeiten die Big-Data-Anwendungen von Google täglich 24 Petabyte an Daten.

Datenselektion findet heutzutage meistens nicht mehr statt, da etwaige zukünftige Analysen diese Daten benötigen könnten. Möglicherweise gehen dadurch  itvolle Informationen verloren. Somit bleiben alle zur Verfügung stehenden Daten existent.

Anstatt relationaler Datenbanken werden spezielle Datenbanken, Frameworks und Architekturen für Big-Data-Anwendungen eingesetzt. Ein Beispiel dafür ist das Framework Hadoop. Dieses verbindet eine Vielzahl von Applikationen zur Bearbeitung von Big-Data-Anwendungen und liefert somit ein hervorragendes Ökosystem.

1.1 Big Data

Der Begriff Big Data bezeichnet zum einen Datenmengen, welche an Größe und Komplexität nicht mehr mit klassischen Methoden der Datenverarbeitung ausgewertet werden können. Zum anderen beschreibt dieser eine Menge an Technologien, die benötigt werden um große Datenmengen verarbeiten zu können. Solche sind zu komplex, um durch ein einzelnes relationales Datenbanksystem effizient bearbeitet zu werden.



Des Weiteren werden in der heutigen Zeit Daten oftmals unabhängig^{von} ihrer derzeitigen Verwendung verwahrt, um etwaige zukünftige Datenanalyse betreiben zu können. Dies erfordert eine enorme Speicherkapazität, welche mit klassischen Serverstrukturen oftmals nicht mehr zu realisieren ist. Unternehmen, die im Umfeld Big-Data agieren, setzen deshalb auf ein verteiltes System, das den heutigen Anforderungen der Datenspeicherung gerecht wird.

Bevor die Big Data Anwendungen und die Big Data Architektur technisch realisiert worden sind, beanspruchten zum Beispiel Filterungen oder eine Zusammenhangsanalyse auf unstrukturierten Daten zu viel Zeit oder waren aufgrund der Hardwarebegrenzung schlichtweg nicht möglich. Dadurch konnten Unternehmen gegebenenfalls wichtige Informationen nicht erheben, welche ihnen einen erheblichen Wettbewerbsvorteil verschafft hätten.

Allerdings wird das Thema Big Data kontrovers diskutiert. Zum einen benötigen die Geheimdienste wie der BND oder die NSA Vorratsdatenspeicherung um effektive Terrorismusabwehr betreiben zu können, zum anderen fürchten die Bürger den Beginn oder das Fortschreiten einer ständigen Überwachung, welches schlussendlich einem Eingriff beziehungsweise zum Verlust der Privatsphäre gleichkommt. Daher sind insbesondere Informatiker dazu aufgerufen den Bürgern das Thema Big Data und Datenschutz näherzubringen, um dem öffentlichen Misstrauenswachstum entgegenzuwirken und die Bevölkerung in diesem Bereich aufzuklären.




2 Aufgabenstellung

In Form einer Beispielanwendung soll der Umgang mit Big Data Anwendungen gezeigt werden. Hierbei wird das zuvor genannte Hadoop-Framework genutzt. Als Beispielanwendung werden Taxifahrten aus New York City verwendet. Es wird analysiert, wie viele Taxen in den einzelnen Bereichen von New York zu gewissen Zeitintervallen Passagiere aufnehmen. Anschließend wird eine Prognose für einzelne Teilgebiete von New York erstellt. Diese bietet Taxiunternehmen die Möglichkeit eine Prognose darüber zu  fen, wie viele Taxen an einer bestimmten Tageszeit in einem bestimmten Gebiet vor Ort sein sollten, um die Anfrage an Taxifahrten der Öffentlichkeit bedienen zu können. 

2.1 Analyse

Für die Analyse der Taxifahrten wird über New York ein feines Raster gelegt. Innerhalb dieser Rasterstruktur werden die Start- und Endpunkte einer Taxifahrt gezählt. Für jeden Monat innerhalb des zur Verfügung stehenden Zeitraums werden dabei die Start- und Endpunkte getrennt voneinander summiert. Das Ergebnis dieser Analyse bildet die Basis der nachfolgenden Prognose. Die Analyse wird mit Hilfe von Hive durchgeführt und im späteren Verlauf dieser Arbeit erläutert. Nachdem die Summe der Start- sowie Endpunkte einer Taxifahrt gebildet wurde, werden diese visuell im Raster dargestellt. Ermöglicht wird dies über ein R-Skript, welches eine Heat-Map von New York erstellt und in Abhängigkeit der Fahrtenanzahl die einzelnen Raster farblich kennzeichnet.

2.2 Prognose

Um die Verteilung der Taxen optimal auf den Bedarf von New York City anzupassen, wird eine Vorhersage erstellt, die aufzeigt, wie viele Taxen in einem Bereich zu einer gewissen Zeit benötigt werden. Dabei ist es möglich auf den Dimensionen Zeit und Raum die Prognosen in ihrer Granularität anzupassen. Damit diese Funktionalität gewährleistet werden kann, muss der zugrundeliegenden Datensatz erweitert werden. Eine Erweiterung der Attributmenge wird allgemein als Feature Extraction bezeichnet. 

Bei Prognosen muss zuerst ein Modell erstellt werden, welches anhand von zur Verfügung stehenden Attributen eine Vorhersage über einen gewünschten Wert trifft. Es müssen geeignete Datensätze in einer sogenannten Trainingsmenge bereit stehen, bei denen das Prognoseergebnis bekannt ist, als auch geeignete Datensätze, mit welcher das Modell getestet werden kann, der sogenannten Testmenge. Die Testmenge enthält kein Prognoseergebnis, sondern muss entweder manuell oder durch automatisierte Verfahren bewertet werden.

Die Trainingsmenge bezieht sich dabei auf die Datensätze vom Beginn des Jahres 2010 bis zum Ende des Jahres 2012. Die Testmenge kann dabei eine beliebige Zeitangabe und ein beliebiger Ort innerhalb New Yorks sein, wobei diese aufgrund des Validierungsvorganges auf das Jahr 2013 beschränkt wurden, sodass Werte aus dem Prognosemodell mit den tatsächlichen Werten des Jahres 2013 verglichen werden können.

3 Technologien

In diesem Abschnitt werden die verschiedenen eingesetzten Technologien beschrieben.

3.1 Hadoop

Dieses Kapitel wird genauer auf das Hadoop-Framework eingehen. Dabei werden die wichtigsten Applikationen beschrieben, die innerhalb des Projektes genutzt werden.

3.1.1 HDFS

Das Hadoop Distributed File System, kurz HDFS, ist das von Hadoop eingesetzte Dateisystem. Dieses wird in Abbildung 1¹ dargestellt und soll im nachfolgenden näher erläutert werden.

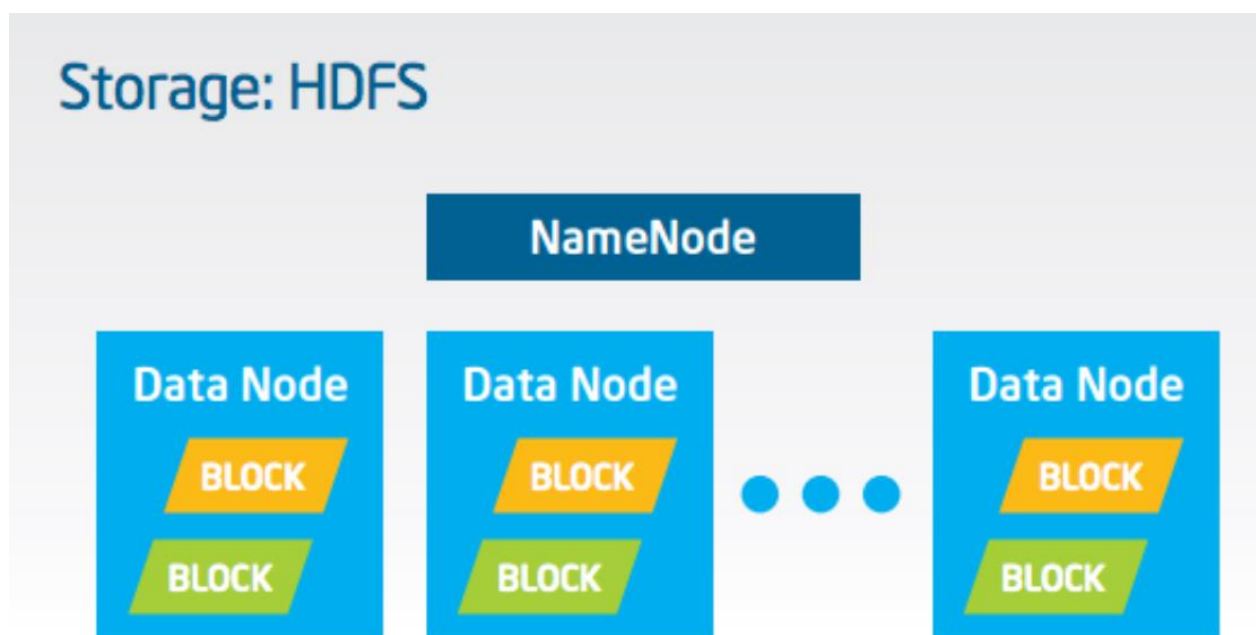


Abbildung 1 – HDFS

HDFS ist ein auf Cluster basierendes Dateisystem, welches sich durch große Flexibilität, Speicherkapazität und verteiltes Arbeiten auszeichnet. Es kann durch Hinzufügen weiterer Elemente im Cluster (Datanodes) ausgebaut und optimiert werden. Jede Datanode stellt einen Teil des Gesamtsystems dar.

¹ Quelle: Intel White Paper: Extract, Transform, and Load Big Data with Apache Hadoop

Die sogenannte Namenode ist die zentrale Komponente des HDFS. Diese speichert alle Metadaten des Dateisystems, kennt alle Datanodes des HDFS und initiiert Schreib- und Leseprozesse.

Zum Zweck der Datensicherheit ist es möglich, weitere Namenodes in das System einzubinden. Diese redundanten Einheiten treten im Fehlerfall stellvertretend für die bisherige Namenode ein und garantieren Ausfallsicherheit. Die Datanodes beinhalten die gespeicherten Daten. Die Daten werden in kleine Blöcke mit einer festen Größe von standardmäßigen 64 MB aufgeteilt. Im realen Betrieb können es aber auch 128 MB oder größere Datenblöcke sein.

Um die Ausfallsicherheit zu erhöhen, wird ein Block, der in einer Datanode gespeichert ist, standardmäßig auf drei andere Datanodes im Cluster repliziert. Daher sollte abgewogen werden, ob der erhöhte Speicherbedarf einer hohen Replikationsanzahl in Kauf genommen werden sollte, oder ob die standardmäßige Replikationsanzahl (3) bereits ausreicht.



3.1.2 MapReduce

MapReduce ist ein Verfahren zum verteilten Bearbeiten von Daten innerhalb von Hadoop. Dieses Verfahren wird von mehreren Applikationen innerhalb von Hadoop verwendet. Es ermöglicht eine effizientere und schnellere Datenverarbeitung, indem parallele Methoden auf verteilt gespeicherten Daten angewendet werden.

MapReduce verarbeitet dabei ^{auch} unstrukturierte Daten, denen kein festes Schema zugeordnet werden kann und besteht grundsätzlich aus zwei Phasen. Die erste Phase ist die Map-Phase. Diese extrahiert aus dem eingehenden Datensatz die relevanten Daten und bildet aus diesen Schlüssel-Wert-Paare.

In der anschließenden Reduce-Phase werden alle Werte desselben Schlüssels zusammengefasst. So kann beispielsweise aus allen Werten eines Schlüssels das Maximum bestimmt werden. Anhand der nachstehenden Abbildung 2² lässt sich ein MapReduce Vorgang nachvollziehen.

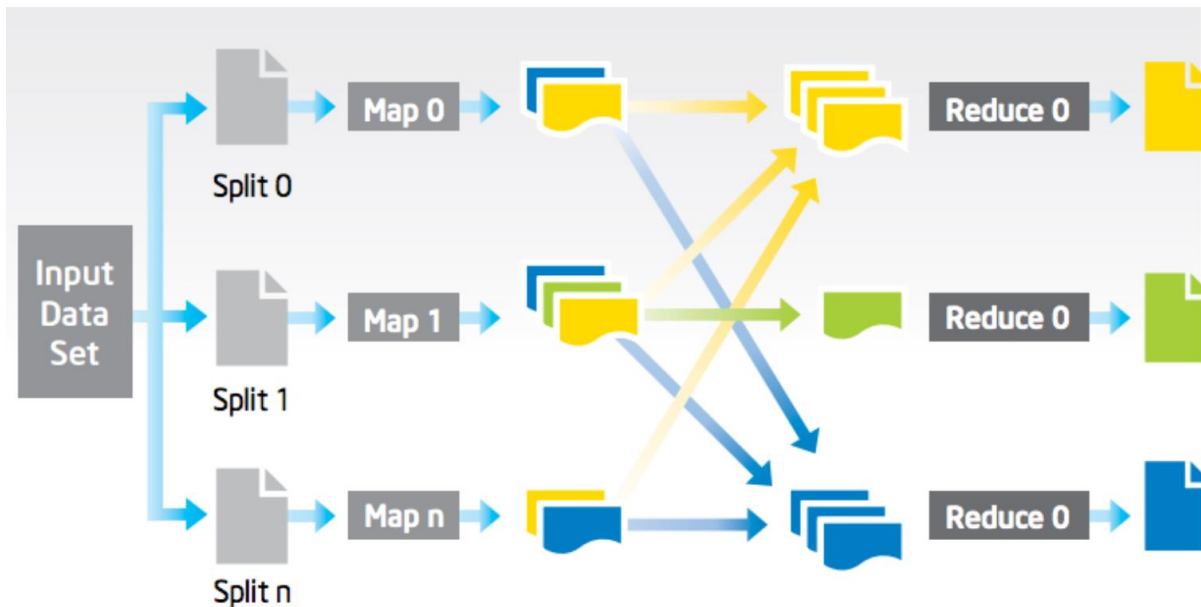


Abbildung 2 - MapReduce

Zuerst werden die Daten aus dem HDFS geladen und durch Mapper verarbeitet. Die resultierenden Ergebnisse werden ihren Schlüsseln entsprechend sortiert und dem jeweiligen Reducer übergeben. Dieser bearbeitet die Werte und überträgt das Endergebnis zurück an das HDFS.

² Quelle: Intel White Paper: Extract, Transform, and Load Big Data with Apache Hadoop

3.1.3 Hive

Apache Hive ist eine weitere Abstraktionsschicht, die Hadoop um eine Data-Warehouse Infrastruktur erweitert, damit das Abfragen, Analysieren und Aggregieren von Daten vereinfacht durchgeführt werden kann. Als Data-Warehouse wird ein großes Datenlager bezeichnet, welches aus unterschiedlichsten Quellen Daten bezieht um diese in einer konsistenten Datenbank bereitzustellen. ETL-Prozesse (Extract, Transform, Load) sollen dabei sicherstellen, dass die Daten möglichst gut aufbereitet (redundanzfrei, konform, gleich strukturiert) sind.

Dabei unterstützt Hive die Analyse großer Datenmengen, beispielsweise im HDFS oder in zu HDFS kompatiblen Dateisystemen. Des Weiteren ermöglicht Hive das Arbeiten mit komprimierten Datensätzen, welches insbesondere im Big-Data Umfeld von hohem Interesse ist, da die Komprimierung von großen Datenmengen erhebliches Einsparungspotential bezüglich Hardwarekosten beinhaltet.

Außerdem können für Hive selbstgeschriebene Funktionen, bekannt als User-Defined-Functions (UDF), verwendet werden. Diese ermöglichen zum Beispiel die Manipulation von Datensätzen oder das Anwenden von Data-Mining Werkzeugen. Hive-Anfragen werden in Form einer SQL-ähnlichen Sprache entgegengenommen, die als Hive-Query-Language bekannt ist (HiveQL). Diese ermöglicht eine wesentlich benutzerfreundlichere Anwendung als beispielsweise das Schreiben eines MapReduce Skripts. Um die im HDFS gespeicherten Daten bearbeiten zu können, werden aus den HiveQL Abfragen MapReduce Jobs erstellt.

3.2 Amazon Web Services (AWS)

AWS ist die Plattform für verteilte Rechendienste von Amazon. Diese bietet viele unterschiedliche Dienste in zehn verschiedenen Kategorien an. Dazu zählen unter anderem virtuelle Server, Online-Speicher, Datenbanken und Analysewerkzeuge für große Datenmengen. Seit der Gründung im Jahre 2006 wird die Plattform ständig erweitert und weltweit neue Rechenzentren eröffnet. Grundsätzlich gilt bei der Bezahlung, dass man nur das bezahlt, was auch verwendet wird.

Im Rahmen dieses Big-Data Projektes wurden folgende AWS Dienste verwendet:

- Simple Storage Server (S3)

Der S3 stellt Online-Speicher zur Verfügung und dient als Datenbasis für viele der anderen AWS Dienste. Dabei erfolgt die Speicherung redundant und auf Wunsch mit Verschlüsselung und Versionierung. Ausgabedateien und Protokolle innerhalb eines Projektes werden in einem sogenannten Bucket Ordner gespeichert. Der Zugriff erfolgt über das Webinterface oder über REST und SOAP kompatible Schnittstellen.

- Elastic Compute Cloud (EC2)

Der EC2 Dienst stellt Serverinstanzen bereit und überwacht deren Ausführung. Es stehen verschiedene Servermodelle zur Verfügung, die innerhalb weniger Minuten über das Webinterface eingerichtet und gestartet werden können.

- Elastic MapReduce (EMR)

Der EMR Dienst ist ein Webservice zum Verarbeiten großer Datenmengen und dient zur Verwaltung von Server-Clustern im Hadoop Ökosystem. Die Ein- und Ausgabe der Daten erfolgt über S3 oder eine AWS Datenbank. Über das Webinterface können Server-Cluster aus EC2 Serverinstanzen angelegt werden. Auf diesen wird automatisch eine Hadoop Distribution installiert, die die meisten der existierenden Hadoop Anwendungen unterstützt. Dazu zählen unter anderem Hive, Pig, HBase und HUE.

3.3 R Programmiersprache

GNU R ist eine Skriptsprache, deren Stärken in der statistischen Auswertung von Daten und deren grafischer Aufbereitung sind. Aufgrund dieser Eigenschaften ist R im Taxi-Trip Projekt genutzt worden, da Teil der Datenanalyse die visuelle Aufbereitung unserer Ergebnisse ist.

Weiterhin bietet R

- Effektive Speicherverwaltung
- Datentypen, die auf statistische Auswertungen zugeschnitten sind
- Operatoren, die auf ganze Datenreihen anwendbar sind
- Ein Programmierparadigma, das die Verarbeitung von Datenreihen stark vereinfacht



4 Daten

Die zugrundeliegende Datenmenge ist von der zuständigen US-Behörde NYC Taxi & Limousine Commission erhoben und mittels FOIL Request beantragt worden. FOIL, die Abkürzung für "Freedom of information laws", sind Rechte welche der Öffentlichkeit ermöglichen auf Daten zuzugreifen, die von der US-Regierung erhoben wurden.

Die Taxi-Daten werden als CSV-Dateien von der US-Regierung bereitgestellt und sind grundsätzlich in zwei Datensätze aufgeteilt:

- trip_data: Angaben über die Taxifahrt (Startpunkt, Dauer, Distanz, ...)
- trip_fare: Aufschlüsselung der Taxifahrtkosten (Trinkgeld, Steuern, ...)

Die Datensätze wurden über folgende Internetseiten heruntergeladen:

- <https://uofi.app.box.com/NYCTaxidata>

Die Datei trip_data_<MONTH>.csv besteht aus folgenden Attributen:

Spalte	Spaltenname	Datentyp	Beschreibung
0	medallion	String	HASH-Wert der Taxi ID
1	hack_license	String	HASH-Wert der Taxi Fahrer Lizenz (vgl. Führerschein) - Fahrer ID
2	vendor_id	String	Hersteller des Taxi-Computers (z.B. CMT, VTS)
3	rate_code	Integer	Taxameter Rate, siehe auch http://www.nyc.gov/html/tlc/html/passenger/taxicab_rate.shtml
4	store_and_fwd_flag	String	store_and_fwd_flag = (Y)es or (N)o,
5	pickup_datetime	datetime	Startzeitpunkt der Taxifahrt
6	dropoff_datetime	datetime	Endzeitpunkt der Taxifahrt
7	passenger_count	Integer	Anzahl der Passagiere/Fahrgäste
8	trip_time_in_secs	Integer	Länge der Fahrt / Fahrzeit in Sekunden gemessen durch Taxameter
9	trip_distance	Float	zurückgelegter Weg in Meilen der Fahrt
10	pickup_longitude	Float	Längengrad beim Start der Fahrt (GPS Koordinaten)
11	pickup_latitude	Float	Breitengrad beim Start der Fahrt (GPS Koordinaten)
12	dropoff_longitude	Float	Längengrad beim Ende der Fahrt (GPS Koordinaten)
13	dropoff_latitude	Float	Breitengrad beim Ende der Fahrt (GPS Koordinaten)

Tabelle 1 - Formatbeschreibung trip_data_<MONTH>.csv

Die Datei trip_fare_<MONTH>.csv besteht aus folgenden Attributen:

Spalte	Spaltenname	Datentyp	Beschreibung
0	medallion	String	HASH-Wert der Taxi ID
1	hack_license	String	HASH-Wert der Taxi Fahrer Lizenz (vgl. Führerschein)
2	vendor_id	String	Hersteller des Taxi-Computers (z.B. CMT, VTS)
3	pickup_datetime	String	Startzeitpunkt der Taxifahrt
4	payment_type	String	Zahlungsart: Bargeld(CSH/CAS), Kreditkarte(CRD/CRE), keine Zahlung(NOC)
5	fare_amount	Float	Reiner Fahrpreis in US-Dollar
6	surcharge	Float	Zuschlag (Rush-Hour oder Nachtzuschlag) in US-Dollar
7	mta_tax	Float	Steuern in US-Dollar
8	tip_amount	Float	Trinkgeld in US-Dollar
9	tolls_amount	Float	Autobahn- bzw. Straßengebühren in US-Dollar
10	total_amount	Float	Gesamtbetrag inkl. Trinkgeld in US-Dollar

Tabelle 2 - Formatbeschreibung trip_fare_<MONTH>.csv

Diese Tabelle wird für das weitere Projekt nicht verwendet, stellt jedoch eine wichtige Datengrundlage für weitere Analysen bereit.

5 Implementierung

In diesem Abschnitt wird die Implementierung des Projektes genauer beschrieben. Dabei werden die Analyse und Prognose einzeln betrachtet.

5.1 Analyse

Wie zuvor dargestellt, wird mit Hilfe eines Hive-Skriptes ein Raster über New York gelegt und innerhalb dieses Rasters die Start- oder Endpunkte der Taxifahrten summiert. Nachstehend wird das Hive-Skript (Abbildung 3) für Endpunkte (dropoff) dargestellt:

```
1 FROM (
2     SELECT
3         ST_Bin(0.001, ST_Point(dropoff_longitude, dropoff_latitude)) bin_id,
4         year(dropoff_datetime) AS y,
5         month(dropoff_datetime) AS m
6     FROM
7         taxi.taxi_data
8     WHERE
9         (dropoff_longitude BETWEEN -74.255 AND -73.7) AND (dropoff_latitude BETWEEN 40.495 AND 40.92)
10 ) bins
11 INSERT OVERWRITE TABLE taxi.data_agg
12 SELECT
13     ST_BinEnvelope(0.001, bin_id) shape, y, m,
14     COUNT(*) count
15 GROUP BY bin_id, y, m;
```

Abbildung 3 - Hive-Skript zur Datenaggregation für die Analyse

Analog dazu wurde das Hive-Skript ausgeführt, welches die Startpunkte (pickup) summiert.

Dieses Skript lässt sich in zwei Teile unterteilen. Der erste Teil besteht aus dem Selektieren vorhandener Daten aus der taxi_data Tabelle. Die vorhandenen Daten beinhalten auch Daten außerhalb von New York City, welche im Projekt nicht analysiert werden. Daher werden nur die Endpunkte ausgewählt, die innerhalb der Grenzen von New York City liegen. Dies geschieht durch die WHERE-Klausel. In dieser wird geprüft, ob der Endpunkt zwischen den Koordinaten der linken oberen und der rechten unteren Ecke von New York City liegt.

Durch die Funktion ST_Bin wird ein Raster mit einer Genauigkeit von 0,001° erstellt. Zusätzlich gibt die Funktion für einen übergebenen Punkt die ID des Quadrats im Raster zurück, in der dieser liegt. In diesem Fall wird für jeden Endpunkt aller Taxifahrten in New York City die zugehörige Quadrat-ID zurückgegeben.

Im zweiten Teil des Skripts werden die selektierten Daten in eine neue Tabelle „data_agg“ zurückgeschrieben. Durch eine GROUP BY-Klausel werden alle Datensätze mit der gleichen ID (also alle Punkte, die in demselben Quadrat des Rasters liegen) zusammengefasst. Damit das Quadrat nachträglich wieder durch Koordinaten abgebildet werden kann, wird die Funktion ST_BinEnvelope verwendet. Diese erstellt anhand der übergebenen ID einen neuen

Datensatz, welcher die vier Eckpunkte des Quadrats enthält. Zusätzlich zu dem Quadrat wird die Anzahl an Taxifahrten, die innerhalb des gleichen Quadrats liegen, in der Tabelle gespeichert.

Innerhalb dieses Hive-Skripts werden sogenannte User-Defined-Functions verwendet. Diese müssen vor ihrer Nutzung in Hive importiert und über die Anweisung „create function“ bekannt gemacht werden (Abbildung 4).

```
1 add jar /home/hadoop/libs/esri-geometry-api-1.2.1.jar;  
2 add jar /home/hadoop/libs/spatial-sdk-hive-1.1.1-SNAPSHOT.jar;  
3 add jar /home/hadoop/libs/spatial-sdk-json-1.1.1-SNAPSHOT.jar;  
4  
5 create function ST_Bin as 'com.esri.hadoop.hive.ST_Bin';  
6 create function ST_Point as 'com.esri.hadoop.hive.ST_Point';  
7 create function ST_BinEnvelope as 'com.esri.hadoop.hive.ST_BinEnvelope';
```

Abbildung 4 - Hive-Skript zur UDF Bekanntmachung

Wie zuvor beschrieben, speichert das Hive-Skript die Daten in eine dafür vorgesehene Tabelle namens „data_agg“. Diese wird über folgende Befehle angelegt:

```
1 CREATE TABLE taxi.data_agg(area BINARY, year INT, month INT, count DOUBLE)  
2 ROW FORMAT SERDE 'com.esri.hadoop.hive.serde.JsonSerde'  
3 STORED AS INPUTFORMAT 'com.esri.json.hadoop.EnclosedJsonInputFormat'  
4 OUTPUTFORMAT 'org.apache.hadoop.hive ql.io.HiveIgnoreKeyTextOutputFormat';
```

Abbildung 5 - Hive-Anweisung zur Erstellung der Analyse-Tabelle

Die Tabelle besitzt die Spalten „area“, „year“, „month“ und „count“. Dabei beinhaltet „area“ die verschiedenen Quadrate des Rasters und „count“ die zugehörigen Summen von Start- oder Endpunkten. Als nächstes wird angegeben wie HIVE die Daten speichern soll.

SerDe ist ein Interface für Input/Output-Operationen. Es gibt einige Standardformate für SerDe, die von Hive genutzt werden um Daten zu serialisieren und zu deserialisieren. In diesem Projekt wird ein benutzerdefiniertes SerDe genutzt. Damit wird Hive mitgeteilt, dass er für die Serialisierung und Deserialisierung „JsonSerde“ nutzen soll.

Die Daten werden nach der Bearbeitung als JSON-Objekt in die zuvor angelegte Tabelle abgelegt. Dabei wird immer ein JSON Objekt pro Zeile verarbeitet. Die Daten kommen als „EnclosedJsonInputFormat“ in Hive an. Bei einem Zugriff auf diese Tabelle werden die Daten als „HiveIgnoreKeyTextOutputFormat“ zurückgegeben. Dies hat zur Folge, dass die Datenausgabe unterbunden wird, indem die Schlüssel zuvor entfernt werden. Somit stehen ausschließlich die aggregierten Daten in der dafür vorgesehenen Tabelle.

Nach der Datenaggregation wird mit Hilfe eines R-Skriptes (siehe Abbildung 6) die geplante Heat-Map gezeichnet. Um die Geodaten, die im JSON-Format vorliegen, verarbeiten zu können, wurde auf zwei Programm-Bibliotheken zurückgegriffen. Zum einen die Bibliothek "json" und zum anderen die für die Verarbeitung und Visualisierung der Geodaten zuständige Bibliothek "RGoogleMaps". Teilaufgabe der Visualisierung ist die Bereitstellung eines Kartenausschnitts, der als Basis der Heat-Map dient. Dies kann mittels "RGoogleMaps" bewerkstelligt werden. Dafür muss der gewünschte Kartenmittelpunkt durch Längen- und Breitengrad definiert und ein Zoomlevel vorgegeben werden.

```

50      # Daten des Jahres und Monats sammeln
51      for (line in 1:length(data)) {
52          if(data[[line]]$attributes$count < threshold)
53              next
54          if(data[[line]]$attributes$year != year)
55              next
56          if(data[[line]]$attributes$month != month)
57              next
58          count <- data[[line]]$attributes$count
59          longitude <- mean(c(data[[line]]$geometry$rings[[1]][[1]][1],data[[line]]$geometry$rings[[1]][[3]][1]))
60          latitude <- mean(c(data[[line]]$geometry$rings[[1]][[1]][2],data[[line]]$geometry$rings[[1]][[3]][2]))
61          lons <- c(lons, longitude)
62          lats <- c(lats, latitude)
63          counts <- c(counts, count)
64      }
65      # Counts logarithmisch machen
66      logc <- log(counts)
67      max_logc <- max(logc) - 2
68      min_logc <- min(logc)
69
70      # Farbvektor erstellen der den Counts entspricht
71      for(c in 1:length(counts)) {
72          colors <- c(colors, colorize(100 * (logc[c] - min_logc) / max_logc))
73      }
74
75      # Plotten
76      PlotOnStaticMap(NY,cex = 0.28, pch = 15, col=colorv, lat=latv, lon=lonv, FUN = points, add=FALSE)

```

Abbildung 6 - R-Skript zur Heat-Map-Generierung

Anschließend werden die aggregierten Daten durch Farbcodes bewertet und gezeichnet. Da der Wertebereich von unter 100 bis 100.000 reicht, wurde die Bewertung logarithmisch vorgenommen.

Nach dem ersten Versuch, die Werte in der Karte darzustellen, wurde folgende “ungenauere” Darstellung generiert.

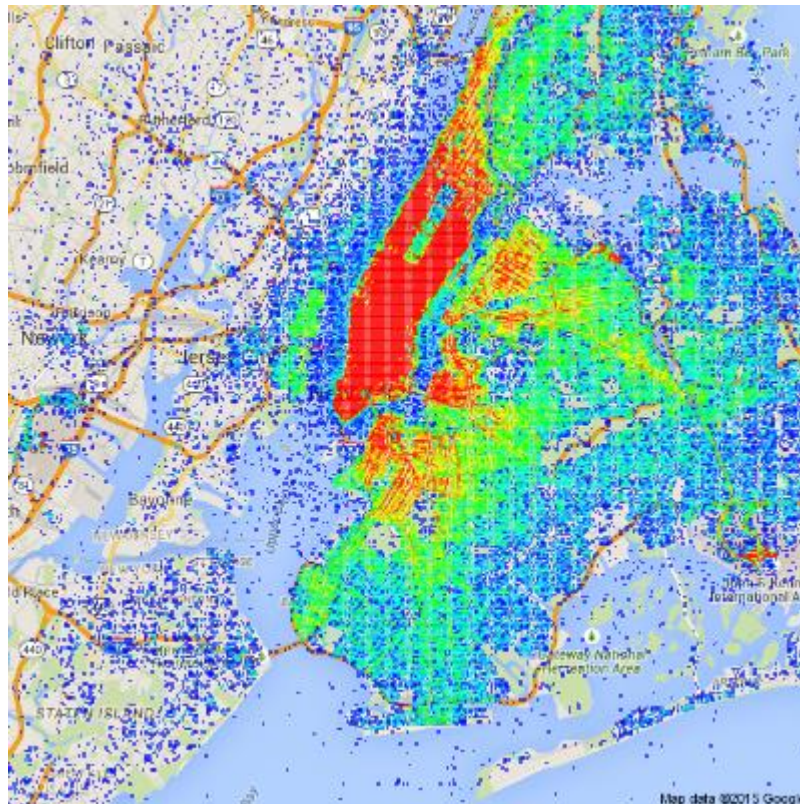


Abbildung 7 - Heat-Map NYC aller Fahrten

Abbildung 7 enthält sämtliche Fahrten, auch Punkte (siehe Meer) die durch GPS-Fehler entstanden sein könnten. Einzelfahrten sind ebenfalls noch enthalten.

Um ein klareres Bild zu erhalten, wurden sämtliche Fahrtziele, die weniger als zehn Mal angefahren wurden, herausgefiltert:

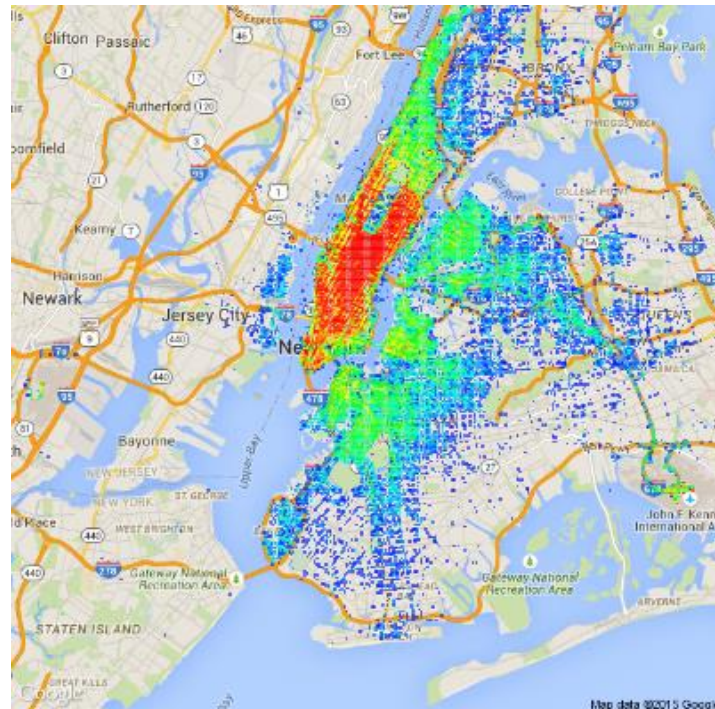


Abbildung 8 - Heat-Map NYC Schwellwert: 10

Das Ergebnis entspricht unseren Vorstellungen. Im Vergleich zu einer, durch die professionelle Software ArcMap erstellten, Heat-Map zeigen sich nur geringe Unterschiede.



Abbildung 9 - Heat-Map NYC der professionellen Software ArcMap

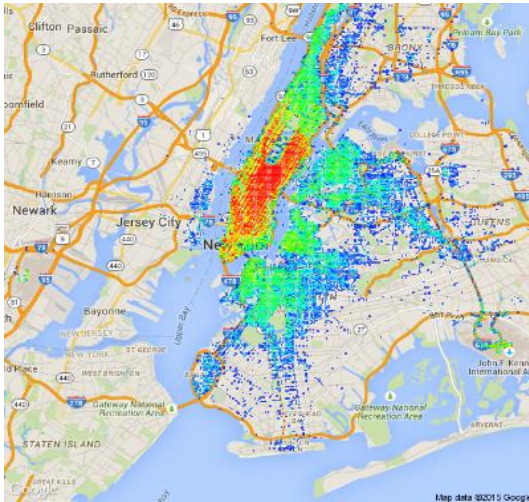


Abbildung 10 - Heat-Map NYC Schwellwert: 15

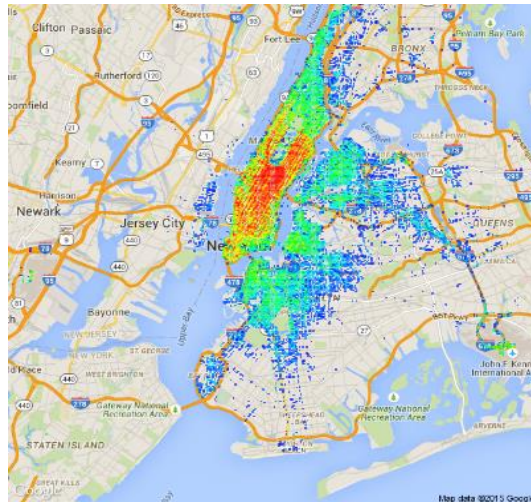


Abbildung 11 - Heat-Map NYC Schwellwert: 20

Abbildung 10 und Abbildung 11 zeigen weitere Heat-Maps mit einer Anpassung des Schwellwertes auf 15 und 20.

5.2 Prognose

Wie zuvor beschrieben soll die voraussichtliche Anzahl benötigter Taxen für einen definierten Bereich in NYC vorhergesagt werden. Dabei gab es folgende Arbeitsschritte:

- Bereichsdefinition
- Selektion der relevanten Fahrten
- Auswahl des Data-Mining-Verfahrens
- Prognoseerstellung



5.2.1 Bereichsdefinition

Die Bereichsdefinition basiert auf den zuvor erstellten Heat-Maps. Bei diesen wurden folgende Bereiche ausgewählt:

- 1. John F. Kennedy Flughafen
- 2. Hell's Kitchen
- 3. Greenpoint
- 4. Rockefeller Center

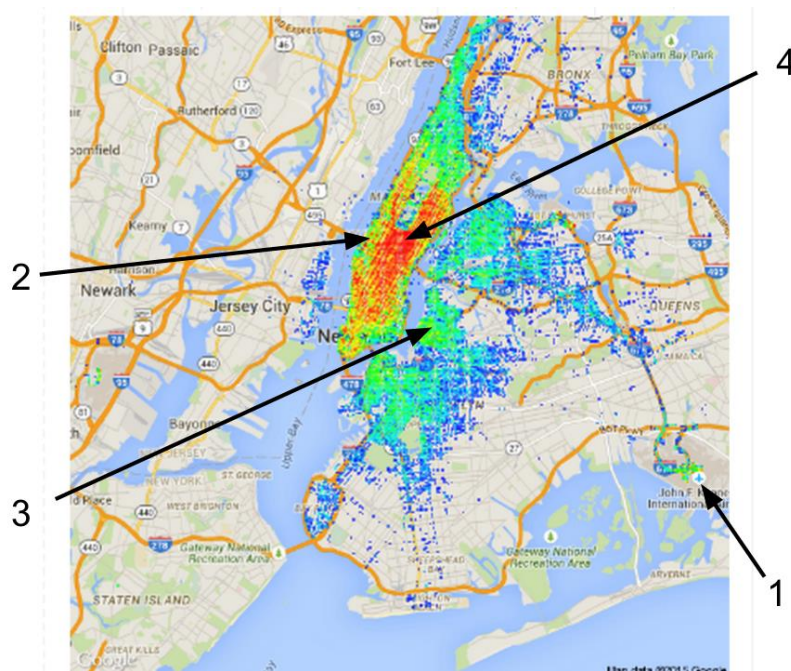


Abbildung 12 - Bereichsdefinition

Abbildung 12 zeigt, dass der John F. Kennedy Flughafen und das Rockefeller Center stark frequentierte Bereiche sind. Sowohl Hell's Kitchen als auch Greenpoint sind weniger stark frequentiert. Durch diese Auswahl ist eine Mischung zwischen mittel und stark frequentierten Bereichen geschaffen worden.

5.2.2 Selektion der relevanten Daten

Damit die Start- beziehungsweise Endpunkte einer Taxifahrt einem Bereich automatisch zugeordnet werden können und nicht umständlich manuell per HiveQL zugewiesen werden müssen, wurde eine eigene User-Defined-Function erstellt. Der Einfachheit halber wurde die User-Defined-Function in einem Maven Projekt erstellt, da durch Einbinden der richtigen Dependencies automatisch alle benötigten jar-Dateien heruntergeladen werden.

Folgende Dependencies mussten dem Projekt hinzugefügt werden:

```
<dependency>
  <groupId>org.apache.hive</groupId>
  <artifactId>hive-exec</artifactId>
  <version>0.13.1</version>
</dependency>
<dependency>
  <groupId>org.apache.hadoop</groupId>
  <artifactId>hadoop-core</artifactId>
  <version>1.2.1</version>
</dependency>
```

Zur Festlegung der Bereiche für die automatische Zuordnung wurde eine eigene Datei angelegt. Dadurch können neue Bereiche hinzugefügt werden, ohne den Quellcode ändern zu müssen. In dieser Datei wird die obere linke und die untere Rechte Ecke einer jeden Zone durch eine eindeutige Zonen-ID identifiziert. Als Format für diese Datei wurde CSV gewählt, da dieses mit Java einfach geparsed werden kann. Die Datei sieht als Tabelle dargestellt folgendermaßen aus:

zoneID	topLeftLat	topLeftLon	botRightLat	botRightLon
1	40.674217	-73.822378	40.640619	-73.772253
2	40.770978	-74.004109	40.754921	-73.986170
3	40.727948	-73.968750	40.701926	-73.928667
4	40.765695	-73.985230	40.753798	-73.972613

Tabelle 3 - Zonenzuordnung

Die User-Defined-Function überprüft nun für einen gegebenen Punkt (definiert durch Longitude und Latitude), ob er sich in einer dieser Zonen befindet. Wenn sich der Punkt zwischen der oberen linken und der unteren rechten Ecke einer Zone befindet, wird die zoneID zurückgegeben. Kann der Punkt keiner Zone zugewiesen werden, wird der Wert -1 zurückgegeben. Durch eine Hive-Abfrage können alle Datensätze verworfen werden, denen keine Zone zugeordnet werden kann (Rückgabewert -1).

Die User-Defined-Function kann durch die nachfolgende Anweisung eingebunden werden:

```
1 CREATE FUNCTION getzone AS 'de.whs.fdt.hive.PointToZoneId'  
2 USING JAR 's3://de.whs.fdb.taxi/libs/FDTTaxiData.jar';
```

Abbildung 13 - Bekanntmachung der eigenen UDF

Dabei wird durch die erste Zeile die Klasse „de.whs.fdt.hive.PointToZoneId“ als Funktion „getzone“ bereitgestellt. In der zweiten Zeile wird der Speicherort der erstellten jar-Datei bekannt gemacht. In diesem Fall liegt die jar-Datei in einem Unterordner des S3-Buckets „de.whs.fdb.taxi“.

Nachdem die User-Defined-Function eingebunden wurde, kann sie beispielsweise folgendermaßen verwendet werden:

```
1 getzone(pickup_latitude, pickup_longitude, '/user/taxi/zonemapping.csv')
```

Abbildung 14 - Verwendung der eigenen UDF

Dabei ist zu beachten, dass der dritte Parameter der Pfad zu der zuvor erwähnten CSV-Datei ist. Es ist zu beachten, dass diese Datei im HDFS liegen sollte.

5.2.3 Auswahl des Data-Mining-Verfahrens

Um ein geeignetes Data-Mining-Verfahren auszuwählen, wurden im Voraus einige Feature-Extractions durchgeführt. Anhand folgender Features wurde das Data-Mining-Verfahren ausgewählt:

- Jahr
- Monat
- Anzahl der Taxifahrten

Testweise wurden sowohl die lineare als auch die logarithmische Regressionsgerade aufgetragen. Die Monatsdaten am John F. Kennedy Flughafen wurden mittels Excel visualisiert. Abbildung 15 zeigt den Verlauf der monatlichen Taxifahrten (pickup) in den Jahren 2010-2013.

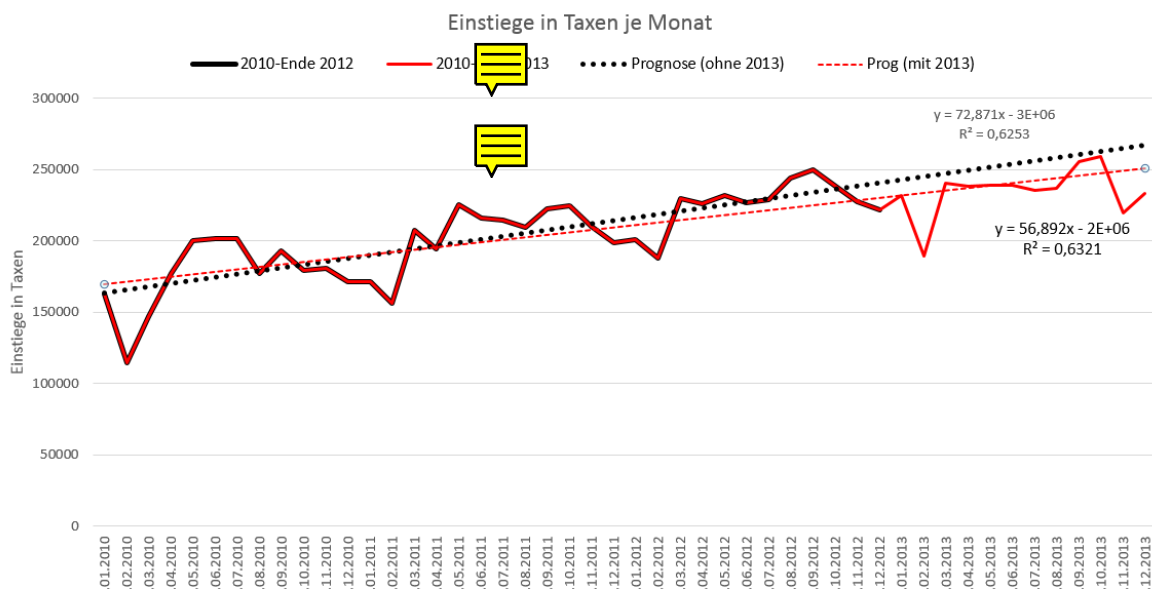


Abbildung 15 - Lineare Regressionsgerade

- Schwarze **lineare** Regressionsgerade
 - Auf Basis des Zeitraums vom 01.01.2010 bis zum 31.12.2012
- Rote **lineare** Regressionsgerade
 - Auf Basis des Zeitraums vom 01.01.2010 bis zum 31.12.2013

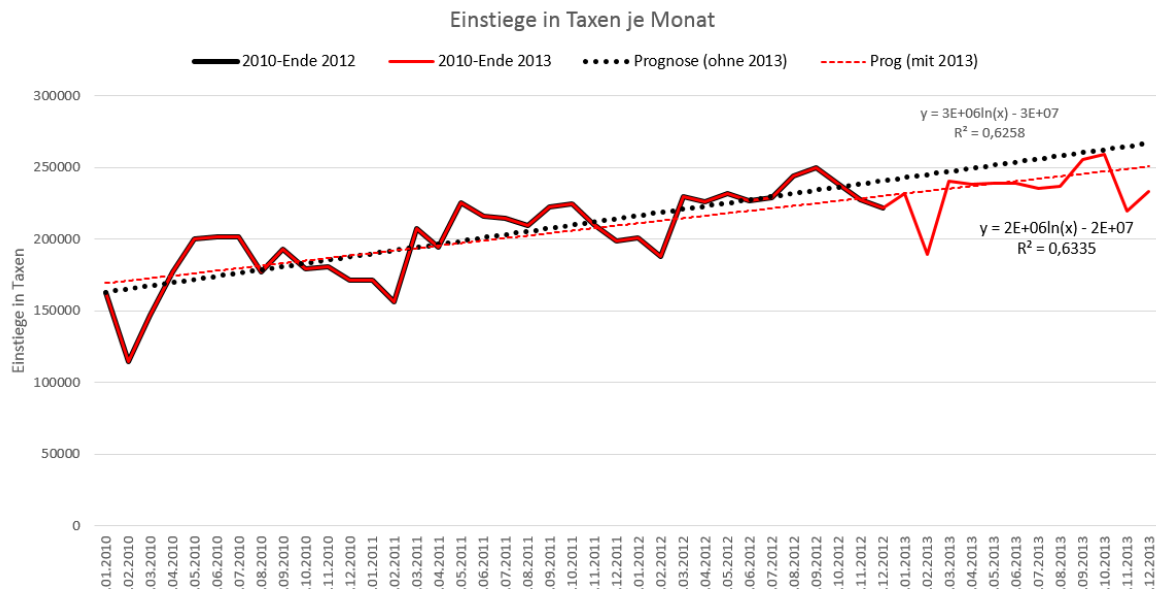


Abbildung 16 - Logarithmische Regressionsgerade

- Schwarze **logarithmische** Regressionsgerade
 - Auf Basis des Zeitraums vom 01.01.2010 bis zum 31.12.2012
- Rote **logarithmische** Regressionsgerade
 - Auf Basis des Zeitraums vom 01.01.2010 bis zum 31.12.2013

Die erhöhte Komplexität der logarithmischen Regressionsgerade brachte gegenüber der linearen Regressionsgerade keinen wesentlichen Mehrwert. Daraufhin wurde bei den nachfolgenden Prognosen stets auf die lineare Regression zurückgegriffen.

5.2.4 Prognoseerstellung – Erster Ansatz

Im ersten Iterationsschritt wurden aus den Ursprungsdaten folgende Attribute (Features) extrahiert:

- Zone
- Anzahl der Taxifahrten
- Jahr
- Monat
- Wochentag

Für die Prognose muss eine Tabelle entsprechend der ausgewählten Features mittels Hive-Anfrage erstellt werden. Diese ist in Abbildung 17 dargestellt.

```
1 CREATE TABLE taxi.data_prog_dayOfWeek (  
2     zone_id INT,  
3     year INT,  
4     month INT,  
5     dayOfWeek INT,  
6     count BIGINT  
7 );
```

Abbildung 17 - Hive-Anfrage zur Erstellung der Prognosetabelle (1. Ansatz)

Abbildung 18 zeigt die Hive-Anfrage, die die zuvor erstellte Tabelle (data_prog_dayOfWeek) mit Daten befüllt. Hier kommt die bereits beschriebene, selbst erstellte User-Defined-Function zum Einsatz.

```
1 FROM (  
2     SELECT  
3         year(pickup_datetime) AS y,  
4         month(pickup_datetime) AS m,  
5         from_unixtime(unix_timestamp(pickup_datetime, 'yyyy-MM-dd hh:mm:ss'), 'u') AS d,  
6         getzone(pickup_latitude, pickup_longitude, '/user/taxi/zonemapping.csv') AS area  
7     FROM taxi.taxi_data  
8 ) area_query  
9 INSERT INTO TABLE taxi.data_prog_dayOfWeek  
10 SELECT  
11     area, y, m, d, count(*) AS count  
12 WHERE area != -1  
13 GROUP BY  
14     area, y, m, d;
```

Abbildung 18 - Hive-Skript zur Aggregation der Daten (1. Ansatz)

Im ersten Teil der Anfrage werden die gewählten Features aus der taxi_data-Tabelle selektiert.

Der nächste Schritt bewirkt, dass alle Daten gruppiert nach Zone, Jahr, Monat und Wochentag in die erstellte Tabelle "data_prog_dayOfWeek" gespeichert werden. Es werden nur Daten aufgenommen, die sich in einem gültigen Bereich (eine der definierten Zonen) befinden.

Die daraus resultierenden Daten wurden erstmals mittels R visualisiert. Die Darstellung (Abbildung 19) zeigt dabei die Anzahl der Taxifahrten auf der Y-Achse und den Zeitverlauf auf der X-Achse (monatsweise).

Ein weißer Punkt stellt die Summe der Taxifahrten der Montage eines Monats dar. Die Menge der weißen Punkte sind alle Monate von 2010-2012 und stellen die Trainingsmenge dar. Der grüne Punkt ist ein Validierungspunkt aus dem zu vorhersagenden Jahr 2013. Die rote Linie entspricht der resultierenden Regressionsgerade, die auf Basis der Trainingsmenge errechnet wurde.

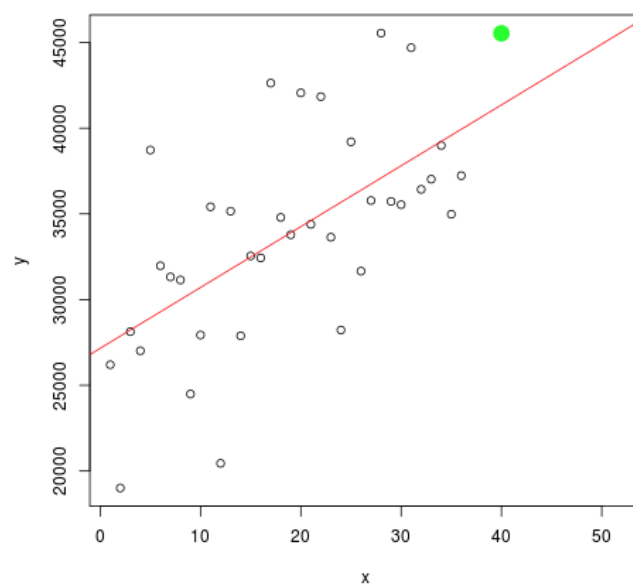


Abbildung 19 - Prognose (1. Ansatz)

Wie in Abbildung 19 zu sehen, ist der Streuungsgrad innerhalb der Trainingsmenge hoch, woraus eine ungenaue Regressionsgrade resultiert. Der Abstand zwischen Vorhersage und Realität beträgt circa 5000 Fahrten. Diese Vorhersage ist für ein Taxiunternehmen unbrauchbar. Daher mussten weitere Feature Extractions durchgeführt werden.

5.2.5 Prognoseerstellung – Zweiter Ansatz

Um die Prognose zu verbessern, wurden die Features auf Folgende erweitert:

- Zone
- Anzahl der Taxifahrten
- Jahr
- Monat
- Kalenderwoche
- Wochentag
- Uhrzeit (Stunde)

Für die erweiterte Prognose muss eine Tabelle entsprechend der oben aufgeführten Features über eine Hive-Anfrage erstellt werden. Diese ist in Abbildung 20 dargestellt.

```
1 CREATE TABLE IF NOT EXISTS taxi.data_prog_final (  
2     zone_id INT,  
3     year INT,  
4     month INT,  
5     weekofyear INT,  
6     dayOfWeek INT,  
7     hourOfDay INT,  
8     count BIGINT  
9 );
```

Abbildung 20 - Hive-Anfrage zur Erstellung der Prognosetabelle (2. Ansatz)

Abbildung 21 zeigt die Hive-Anfrage, die die zuvor erstellte Tabelle (data_prog_final) mit Daten befüllt.

```
1 FROM (  
2     SELECT  
3         getzone(pickup_latitude, pickup_longitude, '/user/taxi/zonemapping.csv') AS area,  
4         year(pickup_datetime) AS y,  
5         month(pickup_datetime) AS m,  
6         weekofyear(pickup_datetime) AS w,  
7         from_unixtime(unix_timestamp(pickup_datetime, 'yyyy-MM-dd hh:mm:ss'), 'u') AS d,  
8         hour(pickup_datetime) AS h  
9     FROM  
10        taxi.taxi_data  
11 ) area_jfk  
12  
13 INSERT INTO  
14     TABLE taxi.data_prog_final  
15 SELECT  
16     area, y, m, w, d, h, count(*) AS count  
17 WHERE  
18     area != -1  
19 GROUP BY  
20     area, y, m, w, d, h;
```

Abbildung 21 - Hive-Skript zur Aggregation der Daten (2. Ansatz)

Analog zur ersten Prognose werden in einem ersten Schritt die Features aus den Daten der Tabelle "taxi_data" extrahiert. Zusätzlich zur ersten Prognose werden die erweiterten

Features berücksichtigt. Im zweiten Schritt werden diese Daten gruppiert in die Tabelle "data_prog_final" geschrieben.

Mittels R wurde abermals die Regressionsgerade erzeugt. Abbildung 22 stellt die Freitage im April 2010-2012 am John F. Kennedy Flughafen von 15-18 Uhr dar. Prognostiziert wurden alle Freitage im April 2013 und durch reale Werte validiert. Zusätzlich wurde neben der Regressionsgerade das Konfidenzintervall berechnet und in die Darstellung aufgenommen. Das Konfidenzintervall ist durch die blau gestrichelte Linie in Abbildung 22 dargestellt.

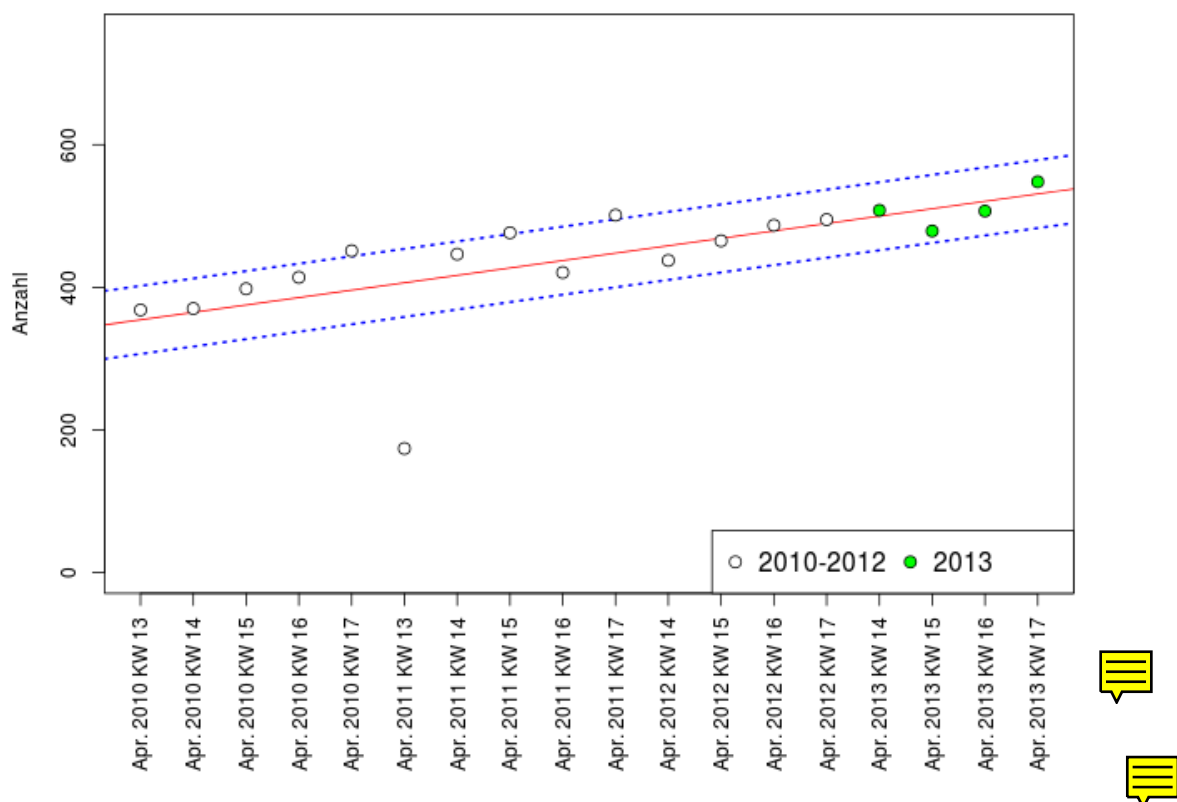


Abbildung 22 – Prognose (2. Ansatz)

Zur Berechnung des Konfidenzintervalls wurden folgende Parameter verwendet:

- Irrtumswahrscheinlichkeit: $\alpha = 0.05$
- Stichprobenmittel: \bar{x} = Wert der Regressionsgerade

Durch die Wahl der Irrtumswahrscheinlichkeit von 0.05 wird dem Taxiunternehmen zugesichert, dass die Anzahl der zu erwartenden Taxifahrten mit einer Wahrscheinlichkeit von 95% im berechneten Konfidenzintervall liegen. Die mathematischen Grundlagen zur Berechnung von Konfidenzintervallen wurde im Modul DSS (Diskrete Mathematik, Stochastik und Simulation) ausführlich behandelt.

Im Gegensatz zur vorherigen Prognose führte die erweiterte Attributmenge zu wesentlich besseren Ergebnissen. Trotz des relativ engen Konfidenzintervalls liegen alle realen Werte des Jahres 2013 innerhalb der Grenzen.

5.2.6 Erkenntnis

Bei der Betrachtung der Taxifahrten in Excel (siehe Abbildung 15) wurde ersichtlich, dass die Anzahl der Taxifahrten im Laufe der Monate steigt. Eine Aufsummierung der Monatswerte am John F. Kennedy Flughafen ergab die folgende Anzahl an Taxifahrten im Jahr:

- 2010: 175708
- 2011: 204419
- 2012: 226194

Durch die Erkenntnis, dass die Anzahl der Taxifahrten steigt, kann ein Taxiunternehmen beispielsweise die Personalplanung verbessern. Außerdem kann im Vorfeld der Kauf weiterer Taxen eingeplant werden um den wachsenden Andrang der Taxinutzung bedienen zu können.

Damit die Taxiunternehmen wissen, in welchem Bereich von New York diese Taxen stationiert werden müssen, wurden in verschiedenen Zeitbereichen und an unterschiedlichen Orten New Yorks detailliertere Prognosen durchgeführt. Die Steigung variiert in den unterschiedlichen Bereichen. Beispielsweise zeigt Abbildung 23 die Prognose für das Rockefeller Center für einen Mittwoch im Januar 2013. Es ist ein leichter Rückgang der Taxifahrten zu erkennen.

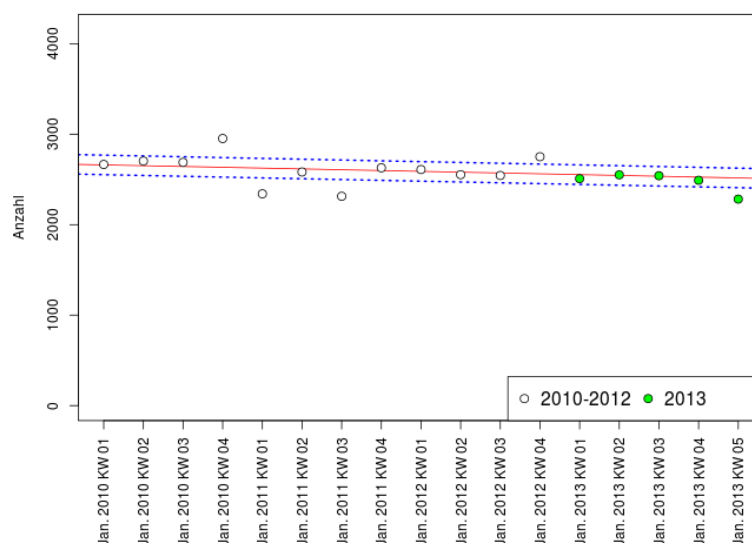


Abbildung 23 - fallender Verlauf der linearen Regressionsgerade

Die Taxiunternehmen müssen daher ihre Taxen insbesondere zu den Orten senden, in denen ein starkes Wachstum der Taxinutzung prognostiziert worden ist.

Das Konfidenzintervall stellt einen weiteren Mehrwert für Taxiunternehmen dar. Es ermöglicht dem Unternehmen Einfluss auf die Auslastung seiner Taxen zu nehmen. Wirtschaftlich bedeutet eine bessere Auslastung der Taxen einen höheren Umsatz bei nahezu gleichbleibenden Ausgaben (Personalkosten, Verwaltungskosten, ...). Dies führt dazu, dass Unternehmen, die über die oben genannten Kenntnisse verfügen, höhere Gewinne erwirtschaften werden.

6 Fazit

Das New York Taxi-Trip Projekt umfasst insgesamt ca. 136 GB Daten, die innerhalb des Hadoop Umfeldes verarbeitet wurden. Dies ermöglichte einen ersten Einblick in das Big-Data Umfeld. Ebenfalls wurde durch diese Datenmengen ersichtlich, warum traditionelle Datenbanksysteme, die auf SQL-Datenbanken beruhen, erhebliche Probleme mit Big-Data haben.

Zuerst wurden einfache Hive-Abfragen abgesetzt, um ein Gefühl für die zu erwartenden Laufzeiten zu bekommen. Die Erstellung der ersten Analysen entstand nach der Einarbeitung in die genutzten Technologien. Dies wurde durch ausreichende Dokumentation und Beispiele seitens Apache ermöglicht.

Für die Erstellung der Heat-Maps stellt R viele Funktionen bereit. Die Nutzung der Bibliothek „RGoogleMaps“ ermöglicht die Einbindung von GoogleMaps-Karten in R. Weiterhin bietet R viele Methoden zur Visualisierung. Einige davon wurden für die Erzeugung der Heat-Map eingesetzt.

Des Weiteren bietet Hive die Möglichkeit User-Defined-Functions als jar-Dateien einzubinden. Durch diese konnte die Zuordnung der Start- und Endpunkte der Taxifahrten innerhalb einer bestimmten Zone im Raster effizient bewerkstelligt werden.

Anhand ausreichender Dokumentation seitens Apache, der im Modul gehaltenen Vorträge, sowie eigener Recherchen über die verschiedenen Technologien konnte eine genaue und vollständige Analyse der New York Taxi-Trips erstellt werden.

Nach der Einarbeitung in Hive konnten die Daten für die Prognose schnell selektiert werden. Hive stellt ein sehr hilfreiches und effizientes Werkzeug dar, mit dem Daten selektiert und in das HDFS gespeichert werden können, benötigt jedoch eine längere Einarbeitungsphase.

Durch die gehaltenen Vorträge im Modul Fortgeschrittene Datenbanktechniken konnte schnell eine geeignete Data-Mining-Methode ausgewählt werden. Um diese durchzuführen, haben wir R verwendet. Auch in diesem Bereich bietet R viele Möglichkeiten und konnte ebenfalls bei der Prognose genutzt werden.

Einzig die Einbindung der User-Defined-Function in Hive war problematisch. Es musste der richtige Pfad angegeben werden, der nicht einfach durch die beigefügte Dokumentation ersichtlich war. Dies kostete während der Entwicklung viel Zeit. Auch die Ausgabe von Fehler-Codes war nicht zufriedenstellend. Hive nutzt dafür einfache Error-Codes. Der geringe Informationsgehalt führt zu einer zeitintensiven Fehlerbehebung.

Mit den erstellten Analysen und Prognosen über die New York Taxi-Trips konnten erste Erfahrungen mit Big-Data-Anwendungen gesammelt werden. Das Hadoop Framework unterstützt Entwickler von Big-Data-Anwendungen und stellt viele hilfreiche Applikationen bereit. In Verbindung mit den AWS und Apache Hive konnten alle Daten der New York Taxi-Trips effektiv und schnell verarbeitet werden. Durch die erstellte User-Defined-Function sowie den R-Skripts konnten die Analysen und Prognosen erstellt und durch externe Software validiert werden.



7 Ausblick

Trotz unserer guten Prognose sind teilweise starke Ausreißer erkennbar (siehe Abbildung 22 – April 2011 KW 13). Um die Prognose weiter zu verbessern, wäre der nächste Schritt die Durchführung einer Ausreißererkennung. Diese kann abweichende Werte erklären oder diese Werte für gewisse Prognosen ausblenden.

Neben der Ausreißererkennung können weitere Datenquellen die Qualität der Prognosen steigern. Wir nehmen an, dass das Wetter einen maßgeblichen Einfluss auf die Taxinutzung hat. Dementsprechend könnte eine Wettervorhersage einen erheblichen Mehrwert darstellen, der bei Verbesserung des Prognosemodells beachtet werden sollte. Weiterhin könnten folgende Datenquellen für weitergehende Auswertungen interessant sein:

- U-Bahn Einstiege / Ausstiege nach Station
- Verkehrsbehinderungen (Baustellen)
- Ortsgebundene Events (Konzerte, Festivals, ...)
- Flughafendaten (Passagieranzahl von Flügen, ...)
- Feiertage

Durch diese Datenquellen kann die Trainingsmenge wesentlich angereichert werden. Die Anreicherung kann vor allem zur Ausreißererkennung genutzt und das Prognosemodell verbessert werden.

8 Anhang

Sämtliche Skripte sind im folgenden Software-Repository zu finden:

- <https://github.com/Giren/taxi>

