DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Analyse und Auswertung von Echtzeit-Fahrplänen der Deutschen Bahn (Project D-Railing)

Studienarbeit

für die Prüfung zum

Bachelor of Engineering

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Alexander Bierenstiel, André Schmitt, Dominik Schmitt

Abgabedatum 14. Mai 2018

Bearbeitungszeitraum	900 Stunden
Matrikelnummer	2496963, 3272367, 7191584
Kurs	TINF15B3
Ausbildungsfirma	Sick AG, E.G.O. Gerätebau, netcup GmbH Waldkirch, Oberderdingen, Karlsruhe
Gutachter der Studienakademie	Prof. Dr. Jürgen Vollmer

Erklärung

Ich versichere hiermit, dass ich meine Studienarbeit mit dem Thema: „Analyse und Auswertung von Echtzeit-Fahrplänen der Deutschen Bahn“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Ort Datum Unterschrift

Sofern von der Ausbildungsstätte ein Sperrvermerk gewünscht wird, ist folgende Formulierung zu verwenden:

Sperrvermerk ja oder nein

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen auerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung der Ausbildungsstätte vorliegt.

Zusammenfassung

Dieses Abstract besser schreiben und eventuell eine englische Übersetzung anfertigen

Die vorliegende Studienarbeit befasst sich mit dem Thema der deutschen Bahn und ihrer Verspätungen. Es soll die von der Bahn zu Verfügung gestellten API genutzt werden, um Daten zu sammeln. Anhand dieser Daten soll ein neuronales Netz modelliert werden, welches genutzt werden kann, um Verspätungen und Abhängigkeiten im Schienen verkehr zu erkennen und vorherzusagen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Quellcodeausschnitte

Abkürzungsverzeichnis

HTTP Hypertext Transfer Protocol 20
BRV Bahnhof-respektive Verzögerung 28
BRVD Bahnhof-respektiver Verzögerungsdurchschnitt 28
SARV Streckenabschnitt-respektive Verzögerung 5

Kapitel 1

Einleitung

1.1 Einleitung

Wie kam es dazu, eventuell mit Motivation kombinieren.

Mit der Bereitstellung der Livefahrplandaten durch die Open Data Bewegung der deutschen bahn[1] kam der Gedanke mit den nun vorhandenen Daten etwas anzufangen. Zuallererst wird eine Diskussion geführt, welche das Ziel der Studienarbeit in der Zukunft bestimmen soll. Aufgrund der Diskussion wurde die Anmeldung und somit die Basis der Studienarbeit geschaffen. Ein Bestandteil der Anmeldung sieht eine Aufbereitung der Rohdaten vor. Hierzu muss eine passende Software entwickelt werden. Die Analyse der Daten soll durch verschiedene Parameter vorgenommen werden, so sollen etwa Verspätungen nach Zeit, Ort oder Strecke betrachtet und ausgewertet werden.

1.2 Motivation

Wieso wollen wir das machen und warum ist das für uns wichtig.

Verspätung im öffentlichen Nah- und Fernverkehr treten täglich auf. Da häufig die Ursachen der Verspätung nicht direkt erkennbar sind, soll mit dieser Studienarbeit die Vorhersage von Verspätung ermöglicht werden. Dafür soll zuerst eine statistische Auswertung der im laufe der Studienarbeit gesammelten Daten durchgeführt werden. Die Auswertung soll die Daten mit Zusammenhang auf ihre Relevanz visualisieren und entsprechend aufbereiten. Dies kann für Pendler von Vorteil sein, um nicht zu spät zu Meetings oder zur Arbeit zu kommen. Ein weiterer gewünschter Nebeneffekt ist die Einsparung unnötiger Wartezeiten auf den gegebenenfalls nächsten Pünktlichen Zug. Durch die gewünschte Erkennung von Regelmäßigkeiten und deren Einflüsse soll die Reisedauer verringert werden.

[1] Siehe URL der DB seite einfügen

1.3 Stand der Technik

Hier etwas zum Stand der Technik schreiben, neuronale Netze, Tensorflow, KI, Datamining, OpenData, etc

Derzeit ist der Begriff: Maschinelles Lernen ein wichtiger Punkt im Fortschritt von Software. In dieser Studienarbeit sollen verschiedene Disziplinen von maschinellem Lernen über Data Mining und Visualisierungstechniken bis hin zur Bereitstellung der Ergebnisse behandelt werden. Es ist wichtig vor beginn der Arbeit die gebiete voneinander abzugrenzen, um die Bearbeitung in kleineren Schritten durchzuführen. Eine gewisse Reihenfolge muss dabei beachtet werden, weshalb im ersten Abschnitt der Studienarbeit auf die Grundlagen eingegangen wird. Zuerst muss der Begriff der Datenbeschaffung und des Data Minings geklärt werden.

Hier Zitat aus Buch Definition zu datamining und datenbeschaffung einfügen

Erst nach der Beschaffung können die Daten in Zusammenhang gebracht werden. Die sinnvolle Visualisierung der Datensätze ist sehr wichtig, um eventuelle Zusammenhänge besser erkennen zu können. Die Visualisierung wird in späteren Kapiteln genauer betrachtet. Eine wichtige Änderung in den letzten Jahren ist der Wille von Großunternehmen einige Daten über eine API Schnittstelle Entwicklern bereitzustellen.

1.4 Ziel der Studienarbeit

Hier das Ziel aus der Anmeldung schön definieren und klar Abgrenzen was Ziel und was optional nice to have ist.

Feststellungen von Verspätungen und Analyse nach

- Ort
- Zeit
- Strecke
- kritische Punkte

Visualisierung der Analyseergebnisse Optional: Vorhersage von weiteren Verspätung durch

- Ort
- Zeit
- Strecke
- kritische Punkte
- Wetterdaten
  - Wind

  - Regen
  - Temperatur

- Höhenlage eines Bahnhofs (z. B. Schneefall)

1.5 Begriffsdefinitionen

Im Rahmen dieser Arbeit werden bestimmte Begriffe verwendet, den eine spezielle Bedeutung beigemessen wird. Damit der Leser diese Begriffe nicht mit der alltäglichen Bedeutung verwechselt, werden sie im Folgenden definiert.

Streckenabschnitt Ein Streckenabschnitt besteht aus einem Gleis oder mehreren Gleise und verbindet zwei Bahnhöfe. Ein Streckenabschnitt wird eindeutig durch die von ihm verbundenen Bahnhöfe identifiziert.

Linie Im Sinne eines Verkehrsnetzes beschreibt die Linie eine Folge von anzufahrenden Bahnhöfen. Um eine Linie eindeutig zu beschreiben, bedarf es einer Menge von Bahnhöfe, die in ihrer anzufahrenden Reihenfolge angeordnet sind.

Kapitel 2

Grundlagen

2.1 Die DB Timetable API

Was bekommen wir eigendlich alles über die Api geliefert

API-URL: `http://api.deutschebahn.com/timetables/v1`
API-Swagger: `https://editor.swagger.io//?_ga=2.234759646.1724072740.1516449724-1264947`
`1510747057#/`

2.1.1 Station

Dieser Endpunkt gibt Informationen über ein Bahnhof zurück. Dafür kann sowohl der Name der Station, die eindeutige EVA Nummer oder die ds100 bzw. rl100 Nummer zur Identifikation angegeben werden. Der Klin'sche Stern kann verwendet werden, um alle Stationen abzurufen. Wurde der Server nicht gefunden, wird der Http-Code **404** zurückgegeben. War der Aufruf erfolgreich, so gibt die API den Status **200** zurück.
Außerdem wird ein Container mit den angefragten Stationen zurückgegeben. Innerhalb eines Stations-Objekt, werden die verschiedenen Identifikationsmöglichkeiten angegeben. Darunter auch die von der Timetable oft genutzte EVA-Nummer. Mit ihr kann jede Bahnstation in Deutschland eindeutig identifiziert werden.
Des Weiteren werden die Plattformen der Bahnstation mit Pipe („|“) angegeben. Der Meta-Eintrag gibt weitere EVA-Nummern an, die mit diesem Bahnhof zusammenhängen (Subbahnhof). Konnte der Bahnhof nicht identifiziert werden, so wird ein leeres Objekt zurückgegeben. Beispiel:
Request:

```
https://api.deutschebahn.com/timetables/v1
/station/Heidelberg%20HBF
```

Response:

```
<stations>
  <station p="4|5" meta="518168|8070043"
  name='Heidelberg Hbf' eva="8000156" ds100="RH"/>
```

```
</stations>
```

2.1.2 Plan

Durch Angabe der EVA nummer (String), eines Datums und einer Stunde, können planmäßige Abfahrten an dem gewählten Bahnhof innerhalb der angegebenen Stunde abgefragt werden. Dabei ist das Datum als String im „YYMMDD“ Format anzugeben. Die Stunde ist ebenfalls als String anzugeben, diese soll im „HH“ Format angegeben werden.

```
/timetable/plan/{evaNo}/{date}{hour}:
    evaNo: Angabe des Bahnhofs
    date: angabe des gesuchten datums (YYMMDD)
    hour: gesuchte stunde (HH)
```

Gibt ein Timetable-Objekt zurück, in dem alle geplanten Abfahrten in der angegebenen Stunde enthält. Dabei werden keine Änderungen durch Verspätungen berücksichtigt.

```
Responses:
200 Successfull operation
```

Gibt ein Timetable-Objekt zurück. In ihm ist der Stationsname, und die EVA-Nummer der Station gekapselt. Außerdem enthält es Listen von Timetable-Stop und Message-Objekten. In einer Plan-Response werden keine Messages übertragen. Es werden nur die "planendÄttribute genutzt.

2.1.3 fchg

Der "fchgËndpunkt nimmt eine EVA-Nummer (String) entgegen und gibt ein Timetable-Objekt zurück. Darin werden alle Änderungen vom Zeitpunkt der Anfrage an gespeichert.

```
/timetable/fchg/{evaNo}:
    evaNo: Angabe des Bahnhofs
```

Innerhalb des Timetabele wird der Name der Station, die EVA Nummer eine Liste von Timetable-Stops und Messages.

2.1.4 rchg

Durch Angabe einer EVA-Nummer können alle Änderungen der letzten zwei Minuten zurückgegeben. Alle 30 Sekunden werden diese aktualisiert.

```
/timetable/rchg/{evaNo}:
    evaNo: Angabe des Bahnhofs
```

Der rchg Endpunkt ist sowohl von den Eingabeparametern als auch von den Ausgabeparametern gleich. der einzige Unterschied ist, dass die Änderungen die Übertragen werden in der Vergangenheit liegen.

Timetablestop In einem Timetablestop werden eine ID aus einer Daily-Trip-ID, Abfahrtsdatum des Zuges am Beginn der Linie und der Nummer des Stops gespeichert. Außerdem die aktuelle EVA-Nummer, die Bezeichnung der Stecke, eine Referenz zum eigentlichen Zug, wenn es ein Ersatzzug ist, die Events Ankunft und Abfahrt, in denen vor allem die An- bzw. Abfahrtszeiten und das Gleis untergebracht sind. Wobei jeweils die geplante als auch die prognostizierte Information enthalten sein kann, eine Massage, warum eine Änderung gemacht worden ist, sowie Informationen, die angeben wie viel Verspätung die Bahn hat und ob sie auf ein anderes Gleis geleitet wurde.

Message Eine Message besteht aus einer Message-Id, einem Message-Typ und einen Timestamp. Des Weiteren können noch folgende Informationen angehängt werden: Eine Information auf welche Uhrzeit der Zug verlegt wurde, aber auch wann der Zug eigentlich geplant war. Ein Code um die Message zu identifizieren, den Text der Nachricht, die Kategorie der Nachricht, die Priorität, der Eigentümer, ein externer Link, der Indikator ob die Nachricht gelöscht ist, eine Nachricht des Verteilers, sowie der Name des Zuges.

2.2 Planung

Zeitliche Einteilung, beachten 5. Semester ist weniger Zeit, Hauptteil wird im 6. Semester passieren

2.2.1 Zeitliche Einteilung der Studienarbeit

Im ersten Teil der Studienarbeit steht die Erfassung der Daten der deutschen bahn im Vordergrund des programmieraspektes. Neben der Programmierung des Data Miners für die Bahn API wird Literatur, welche für die anschließende Aufbereitung und Visualisierung der Datensätze benötigt wird, gelesen. Da die Modulwahl des Teams im fünften Semester eine deutlich höhere zeitliche Belastung durch die Vorlesungseinheiten ausweist, wurde ein Großteil der Hauptarbeit in das sechste Semester verlegt.

Hier Gantt Diagramm oder Tabelle einfügen mit was wurde in welchem Semester gemacht.

2.2.2 Versionsverwaltung

Zur Planung gehört neben der zeitlichen Planung auch die Planung, wie der entstandene Quellcode und die Studienarbeit als Dokument einer Versionsverwaltung unterzogen wird. Die Entscheidung der Gruppe fiel auf Github

eventuell Verlinken

, da damit bereits gute Erfahrungen gemacht wurde. Dort wird eine öffentliche Organisation angelegt, welcher alle Gruppenteilnehmer beitreten. In der Organisation werden die Repositories zur Verwaltung von Website, Data Miner, Visualisierungstoolkit und

Dokumentation angelegt. Alle Teilnehmer bekommen Zugriff auf den Gesamten Quellcode. Damit ist gleichzeitig Backup und ein aktueller Wissensaustausch zwischen den Teilnehmern sichergestellt. Die gemeinsame Arbeit an Quellcode wird durch die Versionsverwaltung erleichtert, da parallel in verschiedenen Branches gearbeitet werden kann.

2.3 Data Mining

Data Mining Einführung und dessen Bedeutung für das Projekt

Data Mining ist ein wichtiger Bestandteil des Projektes, ohne die Daten kann dieses Projekt nicht funktionieren. Denn um ein neuronales Netz zu trainieren, sind Unmengen an Daten nötig. Als Faustregel gilt, je mehr Daten, desto genauer das neuronale Netz. Zum Speichern der Datensätze sollte ein offenes weiterverwendbares Format genutzt werden. Dies soll zudem der weiteren Automatisierbarkeit des Datenflusses dienen.

Datenformat und Aufbau erklären. Wieso sollte im ersten Schritt beim Mining nicht direkt alles angepasst werden? Wieso müssen die Daten aufbereitet werden? Stichwort: FehlerAPI, Fehlende Datensätze, Bucketlist, Konvertierung

Dinge die wir brauchen:

- Bahnhofsnummer
- Linie als Folge von angefahrenen Bahnhöfen (z.B. ICE 690, EC 378, R856)
- Zugreferenz (gleicher Zug auf Linie?)
- Ankunftszeit geplant
- Ankunftszeit real
- Abfahrtszeit geplant
- Abfahrtszeit real
- Historic Delay Element?
  Angeblich kann man damit die vorherigen Verspätungen auf der Linie auslesen
- Wetter je PLZ[Postleitregionen] (Wind, Niederschlag, Temperatur)
- Die Bahnhof Tabelle mit PLZ ergänzen, um Wetterdaten zuordnen zu können (Postleitregionen)

Mögliche Auswertungen:

- Relative Verspätung pro Streckenabschnitt
  Pro Streckenabschnitt kann ein Zug Verspätung aufbauen oder abbauen. Jedem Streckenabschnitt wird die Summe aller Verspätungen, die die Züge auf diesem Streckenabschnitt aufbauen oder abbauen zugeordnet. Diese Summe aller relativen Verspätungen pro Streckenabschnitt wird anschließend visualisiert.

- Verzögerung im Bahnhof
  Pro Bahnhof kann die Verspätung eines Zuges zunehmen oder abnehmen. Pro Bahnhof werden von allen Zügen die Verspätungen, die sie in dem jeweiligen Bahnhof aufbauen oder abbauen, aufsummiert. Anschließend wird für jeden Bahnhof die gebildete Summe visualisiert.

- Welche Wetterlagen bringen Verspätungen

Mögliche Arten der Visualisierung

- Welche Strecken bringen die meiste Verspätung? Heatmap? Top10?

- Welche Bahnhöfe haben die größte Verzögerung? Heatmap? Top 10? Diagramm?

Auswahl der Wetterstationen: Die Wetterstationen werden pro Postleitregion so gewählt, sodass diese möglichst im Zentrum der jeweiligen Region liegen.

2.4 Datenmodell

Datenmodell erläutern, welche Rohdaten aus der DB-API

Ein Datenmodell ist sowohl erforderlich, um Datenobjekte bezüglich ihrer Bedeutung zu interpretieren, als auch, um Beziehungen zwischen Datenobjekten festzustellen oder zu beschreiben. Im Rahmen dieser Arbeit gilt es, ein Datenmodell zu definieren, das unterschiedliche Aufgaben erfüllen soll:

Modellierung realer Größen Zu aller erst definiert das Datenmodell die Modellierung von Größen der realen Welt, die später für die folgende Datenverarbeitung benötigt werden. Hierbei werden mathematische Definitionen entwickelt, die die Bedeutung der jeweiligen Größe, wie zum Beispiel Verspätung, beschreibt.

Modellierung der Rohdaten Anschließend definiert das Datenmodell, wie die beschriebenen Größen der realen Welt in den Rohdaten abstrahiert und abgebildet werden. Dies ist wichtig, um die Rohdaten, wie sie beispielsweise von der Timetable-API der Deutschen Bahn geliefert werden, interpretieren und weiterverarbeiten zu können. Insbesondere muss die Modellierung die Beziehungen unter den Datenobjekten der Rohdaten definieren, um aus diesen wieder die realen Größen ableiten zu können.

Modellierung der Auswertung Nachdem die Bedeutung von realen Größen und deren Abbildung in den Rohdaten definiert ist, muss die Auswertung der Daten konzipiert und modelliert werden. Hierzu zählen sowohl die Beschreibung der internen Darstellung der Daten zum Zwecke der weiteren Auswertung als auch die Beschreibung des auswertenden Algorithmus. Zu den internen Darstellungen können Datenstrukturen in Programmen oder Datenbank-Schemata gezählt werden.

Um die Gliederung der Arbeit übersichtlich zu halten, sind die Modellierungen der oben genannten Punkte in separaten Kapiteln dargestellt.

2.5 Modellierung realer Größen

Schauen, ob Kapitel noch Sinn macht

In diesem Abschnitt werden die realen Größen, die zur Datenauswertung benötigt werden, modelliert. Eine der wichtigsten realen Größen in dieser Arbeit ist die Verspätung oder Verzögerung von Zügen. Im folgenden werden die verschiedenen Arten von Verzögerungen dargestellt.

2.6 Aufbereitung von Daten

Wie werden Daten aufbereitet, vorbereitet für das neuronale Netz, welche Dinge gibt es zu beachten (DATENTYPEN!)

Bei der Aufbereitung der Datensätze geht es die Vorhandenen Daten aufzuteilen, zu kategorisieren, zu formatieren und vorzubereiten. Da im nächsten Schritt ein neuronales Netz trainiert und geprüft werden soll, ist eine Aufteilung der Datensätze in diese drei Kategorien sinnvoll. Später kann dann der Echtzeit Datensatz vorhergesagt werden.

2.7 Neuronalen Netzen an simplen Beispielen erklärt

Kleine Einleitung an einem Simplen Beispiel, Linear Regression oder so. Wieso wir sowas brauchen und weshalb es von Relevanz ist.

Achtung siehe Befriggsdefintionen von enuronalen Netzen, dieses kapitel vllt hier her

2.8 Eingabe- und Ausgabe-Parameter für das Neuronale Netz

Erläuterung welche Informationen in das Neuronale Netz eingegeben werden und welche Daten von dem Netz ausgegeben werden.

Achtung eventuell doppelter Eintrag siehe spätere Kapitel.

Endnutzereingaben: Startbahnhof Zielbahnhof Einsteige-Zeit Zugeingabe (welcher Zug genau?)

Eingabe: Zug-ID Ziel-Bahnhof Um Vorraussagen treffen zu können, braucht das neuronale Netz noch zusätzliche Informationen: Strecke des Zuges? Vergangene Fahrten des Zuges und dessen Verspätung?

Zug-ID

Soll-Ankunftszeit des Zuges
Ausgabe: Voraussichtliche Verspätung in Minuten

2.9 Einrichten der Tensorflow Umgebung

Was wird alles für Tensorflow benötigt

Eventuell how to install tf verlinken

Bevor ein neuronales Netz mit Tensorflow realisiert wird, muss die Umgebung auf den jeweiligen Computern eingerichtet werden. Hier unterscheiden sich die Schritte der Einrichtung je nach Betriebssystem. Unter Windows wird die Einrichtung von Python 3.5.2+ via Installer fertiggestellt. Daraufhin wird mit PIP Installs Packages

Verweis einfügen

das Packet von Tensorflow heruntergeladen und installiert. Daraufhin steht die Grundversion von Tensorflow dem Nutzer bereit. Da Tensorflow vor allem durch eine GPU beschleunigt wird, sollte bei der Verwendung als langfristige Umgebung, die GPU Unterstützung installiert werden. Dies spart vor allem Zeit und somit auch unnötigen Leerlauf beim ausprobieren eines neuen Modells. Unter Linux müssen die Schritte ebenfalls vorgenommen werden, da jedoch die Unterstützung für Linux Server bereits vorhanden ist, wird die Installation vereinfacht und benötigt mehrere Stunden weniger, im Falle einer Fehlersuche. Unter Windows gab es beim einrichten des GPU Support unerwartete Probleme mit den Systemumgebungsvariablen, wodurch die Treiber für die Grafikkarte nicht geladen werden konnten. Da jedoch keine hilfreiche Fehlermeldung erschien, musste die Installation manuell verifiziert werden. Nach diesen Schritten kann Tensorflow mit und ohne GPU Unterstützung auf den Rechnern ausgeführt werden. Ein kurzer Vergleich zeigte, dass die Geschwindigkeit bei Berechnungen mit der Grafikkarte verzehnfacht.

2.10 Literaturhinweise und Empfehlungen

Weiterführende Literatur sollte bis zum Abschluss erwähnt werden, verwendete Quellen zum Einlesen in neuronale Netze und gute Erklärungen, event. Zitate auch benutzen. Diese Autoren sind sehr wichtig für dieses Projekt und sollte auch genannt werden.

Kapitel 3

Datenbeschaffung

3.1 Programmierung des Data Miners

Der Data Miner wird im Laufe der Studienarbeit immer weiter entwickelt und stetig verbessert. Die erste Version zeigte nach nur wenigen Wochen erhebliche Schwachstellen im Quellcode auf. Die erste Version des Data Miners besitzt folgende Funktionen:

- Bahn API aufrufen
- Daten ungeprüft in eine Datenbank schreiben

Durch die geringere Datenmenge (anstatt der 6600 Stationen wurden nur 1200 abgerufen) konnte die Umsetzung schnell realisiert werden. Da es sehr viele Optionen und Probleme gab, wurde die erste Version nach etwa

Anzahl Wochen

Wochen durch die zweite Version des Miners ersetzt. Diese besitzt neben neuen Funktionen auch die Erweiterung der vollständigen Abfrage der API. Die zweite Version konnte die Probleme auf der Seite des Miners minimieren. Die zweite Version kann zudem alle Daten abfragen und nutzt deutlich mehr Informationen, welche in der API der Bahn bereitgestellt werden. Die wichtigste Änderung ist die Fehlererkennung in der Abfrage von Datensätzen. Dadurch soll ein übermäßiges Fehlen von Datensätzen vermieden werden. Die zweite Version des Data Miners ist in der Lage über 600.000 Datensätze am Tag zu verarbeiten. Zu Beginn gab es jedoch noch Probleme mit den aus der API Dokumentation erhalten Datenstrukturen, so sollte ein Gleis angeblich ein Integer sein. Dies trifft jedoch im Falle von "3 A-G", also Gleis 3 Abschnitt A bis G nicht zu. Daher musste die Datenbankspalte für das Gleis angepasst werden. Ebenfalls von Fehlern betroffen war die Zugnummer, diese sollte eine gewisse Länge nicht überschreiten, es gab jedoch Zugnummern mit einer Ziffer zu viel, dadurch konnten Anfangs nicht alle Züge gespeichert werden.

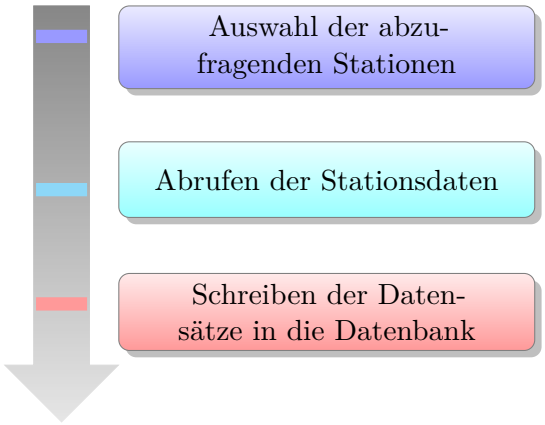

Abbildung 3.1: Grundablauf des Miners

```php
<?php

include_once './settings.php';
require_once 'classes/MysqliDb.php';
require_once 'classes/appgati.php';
// currently not used maybe later
} else {
    $bahnapi = new bahnapi($apikeys2);
}


// Using old querie here because limit dosnt seem to work in rawquery
$params = date("Y-m-d H:i:s", time() - 3600);
$mysqlislave = new mysqli(SETTING_DB_IP, SETTING_DB_USER, SETTING_DB_PASSWORD, SETTING_DB_NAME);

if($minute == 0 || $minute == "00" || $minute == "0") {
    $stationsquery = $mysqlislave->query("UPDATE haltestellen2 set fetchtime='2017-12-01 00:00:00'"); // all stations should be fetched

    // Insert twitter fetch here last 200 tweets lasted over 30 days...


}

$stationsquery = $mysqlislave->query("SELECT EVA_NR as nr, NAME FROM haltestellen2 WHERE fetchactive2=1 AND fetchtime < '$params' ORDER BY fetchtime ASC LIMIT 0,135");

$station = array();
while ($row = $stationsquery->fetch_assoc()) {
```

Quellcode 3.1: Drei Ausschnitte aus einer Datei

Es fehlt eine tabelle zum Vergleichen des Funktionsumfangs der Versionen

Die zweite Version des Miner kann zudem mit den HTTP Status Codes automatisch erkennen, ob es auf der Seite der API gerade ein Problem gibt. So wird auch erkannt, dass es Abends öfter zu kurzen Ausfällen der API mit dem Hypertext Transfer Protocol (HTTP) Statuscode 503

Hier noch was bedeutet 503 und eventuell zitat aus RFC
https://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html

kommt. Dies hilft herauszufinden, ob ein Fehler auf der Seite der BahnAPI oder des Miners vorliegt. Auch ein häufiger Fehler der fehlerhaften initialisierung von Variablen wurde behoben.

Nach der Migration des Miners auf eine größere und schnellere Seite wird die Perfomance der Datenbank erheblich verbessert. Die Datenbank profitiert hier vor allem von deutlich mehr Arbeitsspeicher (anstatt 16 GigaByte nun 64 GigaByte), um Abfragen zwischenzuspeichern. Des weiteren ist der Miner nun IPv6 fähig, da der alte Hostserver noch keine eigene IPv6 Adresse hatte. DIes sichert die Funktionalität im Falle einer IPv6 Umstellung der API Schnittstellen.

3.2 Weatherminer

Wie werden die Datensätze abgespeichert und verwaltet? Das Schema der Datenbank befindet sich im Ressourcen Ordner

Um eine bessere Prognose der Verspätungen zu Ermöglichen, sollte auch das Wetter miteinbezogen werden. Aus diesem Grund wurde entschieden Wetterdaten Deutschlands abzuspeichern um sie Später mit einbeziehen zu können. Dafür wurde eine Datenbankstruktur entwickelt, die das Auslesen der Wetterdaten älterer Daten ermöglicht.

3.2.1 OpenWeatherMap

Durch Angabe der Postleitzahl, Coordinaten oder des Names der Stelle an der das Wetter abgefragt werden soll. In der Antwort können folgende Parameter Ausgelesen werden:

Coordinaten Die Geografische Lage der Stadt die angegeben wurde

Wetter Ein oder mehrere Wetterlagen ID's diese geben Aufschluss über die Wetterlage an dem Ausgewählten Ort. Eine Liste mit allen Wetterlagen sind in der Tabelle 3.1 auf

Seite 22 zu finden. Diese enthält die Wetterlagen ID, die Bezeichnung der Gruppe der Wetterlage, die Beschreibung der Wetterlage und die Kennung des entsprechenden Bild.

Main In diesem Abteil der Antwort werden Werte wie die Temperatur in Kelvin, der Atmospherische druck in hPa, die Luftfeuchtigkeit, die Minimale und die Maximale Temperatur sowie Druck auf Meereshöhe und Druck auf Normalhöhe.

Wind In diesem Abschnitt wird sowohl die Windgeschwindigkeit als auch die richtung des Windes abgelegt

Wolken In diesem Abschnitt wird Abgelegt, Wieviel Prozent des Himmels mit Wolken Bedekt sind.

Regen rain rain.3h Rain volume for the last 3 hours snow snow.3h Snow volume for the last 3 hours dt Time of data calculation, unix, UTC sys sys.type Internal parameter sys.id Internal parameter sys.message Internal parameter sys.country Country code (GB, JP etc.) sys.sunrise Sunrise time, unix, UTC sys.sunset Sunset time, unix, UTC id City ID name City name cod Internal parameter

3.2.2 Datenbank Schema

Für das Datenbankschema wurden die

3.3 Datenbank und Schema

Wie werden die Datensätze abgespeichert und verwaltet? Das Schema der Datenbank befindet sich im Ressourcen Ordner

Ein wichtiger Bestandteil des Projektes ist neben dem Abrufen der API das dauerhafte Abspeichern von Datensätzen. Die Struktur dieser Datensätze hat sich mit der Entwicklung des Data Miners ebenfalls verändert. Es werden mit der zweiten Version deutlich mehr Informationen aus der API abgespeichert. Ein Datensatz benötigt in der ersten Version 140 Bytes und in der zweiten Version 320 Bytes. Viele der neuen Informationen sind für die spätere Arbeit sehr wahrscheinlich wichtig, daher wurden diese in der zweiten Version des Miners ausgewählt. So kann nun der Verlauf eines Zuges besser verfolgt werden und es werden Informationen zum Zugstatus und der Pünktlichkeit strikt getrennt. In Abbildung

x.y

ist das Schema von der ersten Version abgebildet.

Hier etwas darüber erläutern

In Abbildung

x.y

ID	WCondition
200	thunderstorm with light rain
201	thunderstorm with rain
202	thunderstorm with heavy rain
210	light thunderstorm
211	thunderstorm
212	heavy thunderstorm
221	ragged thunderstorm
230	thunderstorm with light drizzle
231	thunderstorm with drizzle
232	thunderstorm with heavy drizzle
300	light intensity drizzle
301	drizzle
302	heavy intensity drizzle
310	light intensity drizzle rain
311	drizzle rain
312	heavy intensity drizzle rain
313	shower rain and drizzle
314	heavy shower rain and drizzle
321	shower drizzle
500	light rain
501	moderate rain
502	heavy intensity rain
503	very heavy rain
504	extreme rain
511	freezing rain

Tabelle 3.1: Tabelle mit allen Wetterverhältnisse

ist dagegen das Schema der zweiten Version zu sehen. Dieses Schema besitzt deutlich mehr Spalten pro Datensatz und benötigt daher auch etwas mehr Speicherplatz. Trotzdem beträgt die Größe der Datenbank nach mehr als 20 Millionen Datensätzen unter 6 Gigabyte. Ein wichtiger Punkt hierbei ist die Menge an Datensätzen. In der Literatur gilt häufig die Faustregel, je mehr Datensätze, desto besser kann das neuronale Netz trainiert werden.

Literatur verweise einfügen

In wie weit diese Aussagen auf dieses Projekt zutreffen wird in Kapitel

x.y

geprüft.

Bei dem Umzug des Data Miners samt Datenbank auf einen neuen Server mussten zehn GigaByte an Datenbank mirgiriert werden. Dies erwies sich als komplizierter als angenommen, denn zum einen Dauert der Export und Import mehrere Stunden und zum anderen müssen die nicht exportierten Einträge des Miners in der Zeit des Umzuges mit dem neuen Server synchronisiert werden. Dies ist bei einer Datenbanktabelle, welche dauerhaft mehrere Transaktionen des Miners bekommt sehr mühsam umzusetzen. Um den Prozess so schonend wie möglich zu machen, wurde ein Skript geschrieben, welches nach der fertigen Migration der Datenbank die Tabellen miteinander Synchronisiert, da ein MySQL Sharding mit Master- und Slave-Modus aufgrund inkompatibler Versionen nicht möglich war. Nachdem das Skript die Tabellen synchronisiert hatte, wurde der alte Miner gestoppt und der Miner auf dem neuen Server gestartet. Die Downtime des Miners betrug nur etwa 60 Sekunden, danach wurde noch einen Fehler in der Installation entdeckt, die durch die Anpassung .

3.4 Backup der Datenbank

Datenkanken sind toll, aber es muss bei eine kritischen Stelle ein backup vorhanden sein.

Jeder Datensatz des data Miners ist wichtig. Daher soll für diese kritische Stelle, der persitenten Speicherung der Daten ein automatisiertes und verifizierbares Backup entstehen. Hierbei gibt es zwei Hauptprobleme zu lösen. Zum einen muss während des Backup eine große Transaktion im Cache oder auf der Festplatte zwischengespeichert werden. Zum anderen ist durch die Menge an Datensätzen eine manuelle Verifikation, ob die Datensätze auch wieder ein spielbar sind sehr aufwändig. Daher wird ein kleines Skript geschrieben, welchen mit linearem Aufwand (Größe der Datei) die Datensätze an bestimmten stellen aufsplittet. So entstehen viele kleinere Dateien. Diese können in unter einer Minute mit einem Datenbank Import auf funktionierende Contraints und geprüft werden. Dies ermöglicht nach einem Vollständigen backup die einzelenen Dateien automatisiert nach und nach in einer kleineren Datenbank zu prüfen. Sollte ein Fehler auftreten, wird dieser in dem MySQL eigenen Fehler Log geschrieben. Hier gilt der Grundsatz, nutzen was schon vorhanden ist. In Listing x.y wird der Quellcode des Skriptes zum aufteilen der Datensätze gezeigt. Die Laufzeit wird grundsätzlich durch die I/O-Geschwindigkeit der Festplatte bestimmt. Die Begrenzung der erstellten Dateien erwies sich bei der Implementierung als Hilfe, um ein fehlerhaftes Anlegen von tausenden kleinen Dateien zu vermeiden.

Listing mit splitfile.php und vllt von Andre das Import script.

Kapitel 4

Datenverarbeitung mit Data Mining

4.1 Grundlagen

- Was ist Data Mining? - Untersuchugsmethoden - Statistik - Maschine Learning - Visualierungsformen

Mögliche Auswerrungen

- Verspätungsarten pro Linie - davon der Durchschnitt über mehrere Züge die die fgeliche Linier an verschiedenen Zeiten befahern - Daraus können Heatmaps ererugt werden

4.2 Vorverarbeitung der Daten

Bevor die gesammelten Daten analysiert werden können, müssen Teile der Datensätze vorverarbeitet werden, um sie in ein brauchbares Datenformat zu bringen.

Strecken eines Zuges werden in langen Zeichenketten statt EVA-Nummern abgelegt

Die Datenbank enthält mehrere tausend Datensätze von Zügen die an verschiedenen Haltestellen und Bahnhöfen halten. Die Strecke, die ein Zug fährt ist eine der wichtigsten Informationen, die aus den Datensätzen herausgelesen werden muss. Jedoch ist das in dem ursprünglichen Format der Datensätze sehr ineffizient auszulesen. Die einzelnen

```
1 import sys
2 from glob import glob
3 import os
4 import pymysql
5 import json
6 import re
7 import io
8     hhmmss = input
9     (h, m, s) = hhmmss.split(':')
10    result = int(h) * 60 + int(m)
11    return result
12
13
14 #used instead of hash buckets to get a better idea of the
      meaing of the values
15 #warning this function is slow
16 def coloumntovocalfileold(name,input):
17    # ii#
18    filename = "./vocabfiles/" + str(name) + ".txt"
19    with io.open(filename, mode="r+", encoding="utf-8") as
         file:
20        for line in file:
21            if input in line:
22                break
23        else: # not found, we are at the eof
24            file.write(input) # append missing data
25
26
27 def openvocalfile(name):
28    # ii#
```

Quellcode 4.1: Some Python File

Ebenfalls für Abfragen ineffizient, ist der "Primary key"der Deutschen Bahn. Dieser besteht , wie in **??** beschrieben aus drei teilen. Um eine Abfrage der Datenbank auf einen bestimmten Zug zu machen muss die Daily Trip ID sowie das Datum im "yymmddhhmm"Format angegeben werden. Diese Werte müssten dann allerdings von der Datenbank mit dem String der Zug Id verglichen werden. Das ist Natürlich nicht sehr performant. Deshalb wurde entschieden, dass die Datenbankstruktur um drei spalten erweitert wird. Da sie im Nachhinein hinzugefügt worden sind müssen alle schon vorhandenen Einträge bearbeitet werden. Dafür wurde ein Algorithmus geschrieben, der die Zugid aus der Datenbank ausließt und wie in 4.2 beschrieben gesplittet. Diese Komponenten werden dann in die jeweiligen Spalten inseriert.

```
1     if temp.empty:
2             print("actual id {} is empty".format(actual_id))
3             missing_IDs.write('{}\n'.format(actual_id))
4             missing_IDs.flush()
5     else:
6     # check if id is already filled
7         if temp["stopid"][0] is None:
8             # split zugid in komponents
9             zugid = temp["ttsid"][0]
10            print("actual id {} \t zugid: {}\n\r".format(
                  actual_id, zugid))
11            zugid = zugid.split("-")
12            # if first komponent is empty dailytripid was
                  negative
13            try:
14                if zugid[0] == "":
15                    zugid[1] = int(zugid[1]) * (-1)
16                    qs.insert_3tuple_with_id(actual_id, zugid[
                          1], zugid[2], zugid[3])
17                else:
18                    qs.insert_3tuple_with_id(actual_id, (zugid
                          [0]), zugid[1], zugid[2])
19            except ConnectionResetError:
```

Quellcode 4.2: Zerlegen der Zug ID in seine Komponenten

4.3 Software-Architektur der Datenauswertung

In diesem Abschnitt wird kurz die Software-Architektur dargestellt, die bei dem Data Mining angewandt wird. Grundsätzlich müssen die Daten zuerst beschafft werden, bevor diese ausgewertet werden können. Aus diesem Grund werden zuerst die nötigen Daten aus der Datenbank abgerufen.

Da je nach Auswertung verschiedene Datensätze aus der Datenbank gebraucht werden, sind auch mehrere SQL-Queries notwendig, um diese Daten von der Datenbank abzufragen. Zu diesem Zwecke werden die Abfragen von der Klasse QuerySuitedurchgeführt. Die Klasse enthält für jede SQL-Abfrage eine spezielle Methode, die diese Abfrage durchführt. Die Abfragen werden in Methoden gekapselt, um das Programmieren übersichtlicher zu gestalten. Jede Methode kann zusätzlich Parameter bei dem Aufruf entgegennehmen, die dann in der SQL-Abfrage verwendet werden können. Auf diese Weise müssen die SQL-Abfragen nicht ständig kopiert und abgeändert werden, wenn sich nur die Parameter, nicht jedoch die Struktur der Abfrage ändern. Diese Maßnahme verhindert die sogennante Code-Duplication, die in der Software-Entwicklung vermieden werden soll. Sollte sich während der Entwicklung beispielsweise das Datenbankmodell ändern, so müssen nur die einzelnen SQL-Abfragen, die in den jeweiligen Methoden der QuerySuite gekapselt sind,

angepasst weden, anstatt nach jeder SQL-Abfrage in jeder Auswertung zu suchen und anpassen zu müssen.

- kümmert sich um die Connnection - LIMIT beschränkung vermeidet versehentlciher Overlosd druch zu grpße abfragen - Bild

ProcessUritl - Enthäkt FUnktionen zum verarbeiten - Enthät mEhtode, die die Verspätung berechnen -

extra Kapitel für Verspätungsberechnungnen - Code beidpiel - erläutern - Gleichungeen

Statistidche Auswerttunfen - Verspätungsarten einer Strecke berechnen - Auswertungen als extra Kaite - Aus Verspätungen können

Query Suite - enthält sql queries - nehmen parameter für die Anfragen entgegen - konvertiert queries in dataframes

4.4 Untersuchung der Verspätungen

Für die Auswertung der Daten ist die Verspätung eine interessante Größe. Hierbei können verschiedene Verspätungen definiert und in dem Datenbestand untersucht werden. In diesem Abschnitt werden verschiedene Verspätungsarten definiert und anschließend dargestellt, wie diese in dem Datenbestand analysiert werden.

Ankunftsverzögerung Die Verzögerung der Ankunft Δan eines Zuges zug_n im Bahnhof bhf_m ist definiert als

$$\Delta an(bhf_m, zug_n) := an_{real}(bhf_m, zug_n) - an_{plan}(bhf_m, zug_n) \tag{4.1}$$

Abfahrtsverzögerung Die Verzögerung der Abfahrt Δab eines Zuges zug_n im Bahnhof bhf_m ist definiert als

$$\Delta ab(bhf_m, zug_n) := ab_{real}(bhf_m, zug_n) - ab_{plan}(bhf_m, zug_n) \tag{4.2}$$

Bahnhof-respektive Verzögerung (BRV)

$$brv(bhf_m, zug_n) := \Delta ab(bhf_m, zug_n) - \Delta an(bhf_m, zug_n) \tag{4.3}$$

Anhand der BRV lässt sich erkennen, ob der Zug Verspätung während dem Verweilen in dem Bahnhof aufbaut. Ist die BRV positiv, so nimmt die Verspätung des Zuges durch die außerplanmäßige verlängerte Haltedauer zu. Ist die BRV hingegen negativ, so verringert sich die Verspätung des Zuges durch eine verkürzte Haltedauer im Bahnhof. Ist $brv = 0$, so entspricht die Haltedauer des Zuges der geplanten Haltedauer.

Bahnhof-respektiver Verzögerungsdurchschnitt (BRVD)

$$brvd(bhf_m, Z) := \sum_{i=0}^{n} \frac{brv(bhf_m, z_i)}{n} \tag{4.4}$$

Der BRVD berechnet den Durchschnitt des BRV bezüglich einer Zugmenge Z. Mithilfe des BRVD lässt sich interpretieren, wie stark die Züge im Durchschnitt durch den Halt in dem jeweiligen Bahnhof verzögert werden.

SARV

$$\begin{aligned} sarv(bhf_{ab}, bhf_{an}, zug_n) := \quad & [an_{real}(bhf_{an}, zug_n) - ab_{real}(bhf_{ab}, zug_n)] - \\ & [an_{plan}(bhf_{an}, zug_n) - ab_{plan}(bhf_{ab,zug_n})] \end{aligned} \tag{4.5}$$

Die SARV erlaubt es festzustellen, ob der Zug auf dem Streckenabschnitt von den letzen Bahnhof zum nächsten Bahnhof Verzögerung aufbaut oder abbaut.

Mit den oben definierten Verzögerungen ist es bereits möglich, erste statistische Auswertungen auszuführen über die Verspätung von Zügen, die sich entweder in Bahnhöfen oder auf den Strecken zwischen Bahnhöfen ereignen.

Implementierung

Die Berechnung der verschiedenen Verspätungen sind mit der Programmiersprache Python implementiert. Der grundsätzliche Ablauf der Auswertung ist wie folgt. Zuerst werden die benötigeten Daten aus der Datenbank abgerufen und anschließend ausgewertet.

```
1 def calc_delay_by_traveltime_df(train_stop_from_df,
      train_stop_to_df):
2     """
3     Calculates the delay that has been caused by the travel of
          the train.
4     Positive value means, that the travel time caused
          additional delay.
5     Negative value means, that the travel time decreased the
          delay.
6     :param train_stop_from_df: Pandas dataframe. Input for the
          train stop the train comes from.
7     :param train_stop_to_df: Pandas dataframe. Input for the
          train stop the train arrives at.
8     :return: Returns a pandas dataframe with columns '
          delay_by_traveltime', 'ttsid_from', 'ttsid_to'.
9     """
10    if train_stop_from_df is None:
11        ttsid_from = None
12    else:
13        ttsid_from = train_stop_from_df["ttsid"].iloc[0]
14
15    if train_stop_to_df is None:
16        ttsid_to = None
17    else:
18        ttsid_to = train_stop_to_df["ttsid"].iloc[0]
19
20    if train_stop_from_df is None or train_stop_to_df is None:
21        delay = pd.NaT
22    else:
23        traveltime_real = calc_traveltime_real_df(
              train_stop_from_df, train_stop_to_df)["
              traveltime_real"].iloc[0]
24        traveltime_scheduled = calc_traveltime_scheduled_df(
              train_stop_from_df, train_stop_to_df)["
              traveltime_scheduled"].iloc[0]
25        delay = traveltime_real - traveltime_scheduled
26
27    result = pd.DataFrame(
28        data=[[delay, ttsid_from, ttsid_to]],
29        columns=["delay_by_traveltime", "ttsid_from", "
              ttsid_to"])
30    return result
```

Quellcode 4.3: Berechnung der SARV

4.5 Stochastische Analyse

Viele der Informationen können auch ohne Neuronale Netze gewonnen werden. Beispielsweise kann aus der wirklichen Abfahrtszeit des vorherigen Bahnhofes und der wirklichen Ankunftszeit des Nächsten Bahnhofes die Fahrzeit der Züge berechnet werden. Des Weiteren kann durch die Ankunftszeit und der Abfahrtszeit des Zuges auch die zeit bestimmt werden, die der Zug benötigt, um eine bestimmte Strecke zu fahren.

4.5.1 Analyse eines Zuges

4.5.2 Durchschnitt der Zeiten einer Strecke

Um die Durchschnitte der Fahrten zu berechnen, wird zu beginn jede strecke einmal Analysiert (siehe 4.5.1). Die Daraus Resultierenden Daten werden dann Bahnhof für Bahnhof Summiert und danach ,wie für einen Durchschnitt nötig, mit der Anzahl der Summanden geteilt. Problematisch sind vor allem Züge die Frühzeitig ihre Fahrt beenden, da diese dann natürlich einige Bahnhöfe nicht anfahren. Das kann dazu führen, dass bei der Berechnung der Durchschnitte Fehler passieren. Oder noch schlimmer das Ergebnis verfälschen.

4.6 Visualisierung

Kapitel 5

Datenverarbeitung mit neuronalem Netz

5.1 Programmierung der Automatischen Datenverarbeitung

Wie kommen die Datensätze aus der DB zum neuronalen Netzwerk, wie wird die Formatierung vorgenommen

5.2 Vorverarbeitung der Datensätze

Kurze Einführung schreiben

id Id als Primärschlüssel zur Speicherung in der Datenbank.

zugid Beispiel: **-7714364757423921343-1712081222-8**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugverkehrstyp Beispiel: **F**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugtyp Beispiel: **p**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugowner Beispiel: **80**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugklasse Beispiel: **ICE**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugnummer Beispiel: **788**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugnummerfull Beispiel: **ICE788**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

linie Beispiel: –**leerer String**–
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

evanr Beispiel: **8000152**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

arzeitsoll Beispiel: **16:32:00**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

arzeitist Beispiel: **16:33:00**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

dpzeitsoll Beispiel: **16:36:00**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

dpzeitist Beispiel: **16:38:00**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

gleissoll Beispiel: **7**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

gleisist Beispiel: **7**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

datum Beispiel: **2017-12-08**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

streckengeplanthash Beispiel: **4d0bc383**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

streckenchangedhash Beispiel: **bd84c25a**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

zugstatus Beispiel: **n**
Zug-Id die von der Timetable-API zur Zugidentifizierung genutzt wird.

Hier die generate_csv.py beschreiben

Die Datensätze aus der Datenbank müssen vor dem Import in das neuronale Netz vor verarbeitet werden. Da Tensoren nur aus numerischen Typen bestehen sollten. Hierfür werden kombinierte Datentypen getrennt und in einen passenden Zieltypen konvertiert. Bei manchen Typen ist es jedoch besser, die Möglichen Werte in ein Vocabfile zu schreiben, um diese mit einer n*n Identitätsmatrix im Tensor darzustellen. Alle Datensätze sollen bevor sie in die einzelnen CSV Dateien geschrieben werden in ein einheitliches Format gebracht werden. Ziel ist es Training, Test und Prediction in einem Schritt umzusetzen.

Datum	Beispiel	Datenbank Datentyp	Konvertierter Datentyp (Python)
id	4092195	VARCHAR	
zugid	-7714364757423921343-1712081222-8	VARCHAR	
zugverkehrstyp	F	VARCHAR	
zugtyp	p	VARCHAR	
zugowner	80	VARCHAR	
String zugklasse	ICE	VARCHAR	
String zugnummer	788	VARCHAR	
zugnummerfull	ICE788	VARCHAR	
linie	#leerer String#	VARCHAR	
String evanr	8000152	INT	
arzeitsoll	16:32:00	TIME	
IntType arzeitist	16:33:00	TIME	
IntType dpzeitsoll	16:36:00	TIME	
IntType dpzeitist	16:38:00	TIME	
IntType gleissoll	7	VARCHAR	
String gleisist	7	VARCHAR	
String datum	2017-12-08	DATE	
String streckengeplanthash	4d0bc383	VARCHAR	
streckenchangedhash	bd84c25a	VARCHAR	
zugstatus	n	VARCHAR	

Tabelle 5.1: Vorverarbeitung der Datenbank-Daten

5.3 Begriffsdefinitionen für ein neuronales Netz

Welche Begriffe werden häufig verwendet, sollte man gehört haben und zuordnen können

Beim Einstieg in das Themengebiet neuronale Netze fallen viele fremde Begriffe. Diese sollten vorab geklärt sein, um missverständnisse zu vermeiden. In folgender Auflistung werden die allerwichtigsten Begriffe erklärt, weiter Begriffe können im Internet (siehe Quellenangaben) nachgelesen werden.

Die Liste vervollständigen und eventuell Quelle mit weiterführenden Definitionen angeben

Feature wird ein Attribut einer Zeile genannt, in diesem Fall zählt zum Beispiel die evanr als Feature in dem Datensatz.

Label wird als die Spalte des Datensatzes definiert, welche am Ende vom neuronalen Netz vorhergesagt werden soll. In unserem Fall wäre die Ankunftszeit (IST) eine solche Spalte.

Layer beschreibt eine Schicht von Neuronen, die Anzahl der Neuronen eines Layers wird anhand der sogenannten Hidden Units festgelegt. Diese gibt gleichzeitig die Anzahl der Layer vor. Ein Beispiel: [20,5,10] bedeutet 20 Neuronen im ersten Layer, fünf Neuronen im zweiten Layer und zehn Neuronen im Output Layer

Loss

Accuracy

Optimizer

Estimator

Input Function nennt man die Funktion, welche für die Eingabe von Datensätzen im Training, Testen und Vorhersagen verwendet wird. Die Funktion liest die Datensätze auf der Festplatte ein (zum Beispiel eine .csv-Datei) und gibt zwei Tensoren zurück. Der erste Tensor beinhaltet alle Feature Spalten der Datensätze und der zweite Tensor die Label der Datensätze.

Model Function

Activation Function

Dropout ist ein Float Wert zwischen 0.0 und 1.0, wobei 0.0 für keine fehlenden Verbindungen zwischen den Layern der Neuronen steht und 1.0 bedeuten würde, dass es keine Verbindungen gäbe. Ein guter Wer liegt zwischen 0.0 (ein sogenanntes "fully connected neuronal networköder 0.3). Der Dropout verhindert, dass alle Datenwerte direkt von Relevanz sind und vermeidet somit ein sogenanntes Overfitting des Modells auf die Trainingsdatensätze.

Tensor

Epochs ist die Anzahl an Epochen, welche das Modell durchlaufen soll.

Steps ist die Anzahl der Schritte, die pro Epoche von dem Modell trainiert werden soll. Bei einer Vorhersage wird die Schrittzahl auf die Anzahl der eingegebenen Datensätze gesetzt beziehungsweise automatisch von Tensorflow erkannt.

5.4 Eingabe der Datensätze in Tensorflow

Input Funktion beschreiben

Die Eingabe von Datensätzen und die Vorverarbeitung sind bei der Erstellung eines neuronalen Netzes von hoher Bedeutung. Die Zeit, eine gut funktionierende und schnelle Eingabefunktion zu schreiben macht sich beim Trainieren des neuronalen Netzes bemerkt. Da beim Training viele Datensätze in kurzer Zeit benötigt werden, muss ein Engpass an dieser stelle wenn möglich vermieden werden. Bevor die Eingabefunktion geschrieben wird, müssen die Spalten der Datensätze im Modell angelegt werden. Es wird also ein Modell mit den Spalten als Variablen angelegt, in welches zu späterer Zeit von der Eingabefunktion echte Werte eingesetzt werden. Deshalb befinden sich in einem Modell des neuronalen Netzes auch niemals echte Datensätze sondern nur die Parameter, welche durch das Trainieren erstellt wurden.

Hier ein Code Snippet der inputfn anzeigen und beschreiben

5.5 Anlernen des Netzes

Beim anlernen eines neuronalen Netzes sind sich viele der Quellen einig.

ein paar quellen querverweise/belege hier einfügen

Je mehr Daten vorhanden zum anlernen, desto genauer das daraus entstehenden Modell und somit die Resultate. In diesem Fall sollte dies sich ebenfalls so verhalten. Da jedoch bei der Einarbeitung in Tensorflow und dessen Verwendung sehr viel Zeit geflossen ist, kann diese These nicht belegt werden. Aus Zeitmangel beim Trainieren des neuronalen Netzes auf eigener Hardware muss auf ein Großteil der Datensätze aus Zeitgründen Vorerst verzichtet werden. Vorab gilt es die Genauigkeit auf einer kleinen Testregion zu testen und verifizieren. Das diese Region nicht die Situationen in ganz Deutschland widerspiegeln kann ist im vor hinein klar. Nichtsdestotrotz soll eine Vorhersage im kleinen Rahmen ermöglicht werden. Die Weiterentwicklung des neuronalen Netzes muss in der Zeit nach dem Abschluss verschoben werden oder von einer Gruppe Studenten aus dem folgenden Jahr übernommen werden, da hierfür schlicht und ergreifend die Ressourcen zu knapp sind. Trotzdem soll eine Vorhersage möglich sein mit dem Wissen, dass diese jedoch nicht perfekt sein wird.

Welche Datensätze werden zum Anlernen verwendet, weshalb ist es wichtig nie alle zu nehmen im Bezug auf Test, Predict und welche Verhältnisse sind bei uns sinnvoll anzusetzen

Aufzeigen wie sich die Menge an Daten auf die Genauigkeit auswirkt

Welche Optionen und Parameter können optimiert werden, wie ändert sich dadurch das Ergebnis.

Hier Tabellen mit Vergleich der Methoden und Genauigkeit, Geschwindigkeit, Erläuterungen weshalb das Ergebnis so ist.

5.6 Verifizieren des Netzes

Testen des neuronalen Netzes, Verifikation der Genauigkeit und deren Steigerung durch Training oder Anpassungen des Netzes

5.7 Vorhersagen anhand des Netzes

Vorhersagen aus Daten treffen und anschauen wie gut sie sind, wo gibt es Probleme, welche Probleme treten auf.

Die Vorhersage mit neuronalen Netzen unterliegen einer Grundlegenden Struktur. Durch die Input Funktion werden die bekannten Größen des Modells an Tensorflow gegeben. Dort wird die Vorhersage durchgeführt und liefert einen Tensor als Antwort zurück. In diesem Falle besitzt der Tensor 1441 Klassen, welche jeweils eine Uhrzeit darstellen. Jeder Uhrzeit wird über eine Softmax Funktion eine relative Wahrscheinlichkeit zugeordnet. Dies bedeutet im Klartext, das die Summe aller Klassen gleich 100% entsprechen. Ein erwartetes Ergebnis einer Vorhersage wäre also eine Normalverteilung über einen bestimmten Wert. Dies hätte die Bedeutung, dass ein Zug zum Beispiel mit der Wahrscheinlichkeit 75% genau zu dieser Zeit kommt oder mit 95% Wahrscheinlichkeit in einer Zeitspanne von fünf Minuten um diesen Wert. Je nachdem, wie genau das Modell die Realität vorhersagen kann, kann diese Kurve schmäler werden, wodurch die Wahrscheinlichkeit einer genaueren Vorhersage größer ist.

Hier ein vergleichsgraphen erwartete Verteilung, echte verteilung zeigen.

5.8 Auswertung und Fehlerbehandlung

Was passiert im Fehlerfall, wie erkennt man Fehler, müssen wir Fehler erkennen oder sind Fehler ëgal", wie stellen wir eine GUI bereit, um anderen Menschen die Ergebnisse zu testen, genauere Statistiken zu Zügen je nach Strecke, Uhrzeit etc., vlt. Visuelle Darstellung wie bei Travic oder mit eigenen Heatmaps bzw. Openstreetmap.

Kapitel 6

Visualisierung und Bereitstellung der Daten im Internet

6.1 Aufbau der Website

Wie wird die Website bereitgestellt, was kann sie und welche Views existieren für die Nutzer

Heutzutage ist die Bereitstellung einer Website eine einfache Methode Daten mit anderen Menschen zu teilen. Da unsere Datenbank und Website einen gewissen Sicherheitsstandard erfüllen soll, wird sich für ein Framework entschieden, welches bereits integrierte Sicherheitsfunktionen bietet. Der name des Frameworks lautet Laravel

Hier reflink zu laravel seite einfügen

. Dies spart vor allem Zeit bei der Entwicklung der neuen FUnktionen für die Bereitstellung der einzelenen Webviews. EIn View ist eine Seite oder der Teil einer Website, welcher in eine weitere Seite eingebettet sein kann. In Abbildung x.y kann die Struktur der einzelnen Ansichten erkannt werden.

Hier Abbildung View mit mehreren Subviews

Auf der Website gibt es folgende Hauptpunkte, welche jedem Nutzer zur verfügung stehen.4

Home ist die Startseite der Nutzer. Hier sollen Grundinformationen an die Nutzer gegeben werden, wie zum Beispiel die Anzahl an Datensätzen (gesamt). Diese Seite soll optimiert sein schnell zu laden, weshalb sie relativ wenig Daten an den Nutzer senden soll. Dies ist vor allem in anbetracht auf die Moile Nutzung der Website wichtig.

Toplist Hier die Punkte der Website updaten wenn sich etwas ändert.

Map ist eine Karte, welche als Basisoberfläche Kartenmaterial von OpenStreetmaps.org verwendet. Darauf werden mithilfe von leaflet einzelnen Schichten gezeichnet, wie zum beispiel die Bahnhöfe und die Streckenverläufe der deutschen bahn.

Hier links zu OSM und leaflet einbinden, sowie zu den rohdaten von db bzw. dem github repo

Stationen ist die Hauptansicht für Statisten der einzelnen Stationen. Auf der Hauptseite befindet sich eine Suchfunktion mit grundlegenden Einstellungen. Nach erfolgreicher Suche nach einem Bahnhof kann sich der Nutzer eine der vielen erzeugten Ansichten anschauen.

Impressum ist eine Verlinkung auf das nach deutschem Recht benötigte Impressum einer Website gemäß § 5 Telemediengesetz (TMG)

6.2 Erstellung der Webrouten

In Mittlerweile fast allen Webframeworks gibt es eine native Unterstützung für Restful basierte Routen. In Laravel werden hier die Routen nochmals in vier Kategorien je nach Anwendungsfeld aufgeteilt. Diese Routen sind nach deren Zuständigkeit benannt und heißen api, channels, console und web. Im Normalfall reichen die Webrouten für das vorhaben aus, falls es eine komplette API für alle Datenätze geben soll, kann diese über die API Routen definiert werden. Der Aufbau einer Webroute ist relativ simple, wie in Listing x.y zu sehen ist.

Listing einer Webroute

. Zuerst wird die Route ausgehend vom Startpunkt der Webseite angegeben. In der Route können Parameter mit geschweiften Klammer als Platzhalter dargestellt werden. So ist es möglich ROuten für alle Stationen anzulegen, ohne diese einzeln Programmieren zu müssen. In der Route wird dann der Parameter aus der URL genommen und anhand dessen der Inhalt der entsprechenden Seite angezeigt.

Beim erstellen der Routen gibt es jedoch auch Fallstricke, welche zu Beginn nicht direkt erkenntlich sind. So muss zum Beispiel die längste Route zuerst angegeben werden, da die Routen nach dem First Match prinzip abgearbeitet werden. Sollte unter der Hauptroute noch eine Subroute mit Parameter stehen wird trotz Parameter nur die Hauptroute angezeigt.

Event Abbildung wie die Reihenfolge richtig und wie falsch aussieht als Listing

6.3 Erstellung der Seiten

Hier was zu Mockups und usability einbringen mit Beispielen anhand der Website

Bei der erstellung der einzelnen Ansichten der Website wird zuvor eine grundlegende Strukturierung anhand von Mockups erstellt. Diese dienen dazu schnell Änderungen vorzunehmen und diese anschließend nach verschiedenen Faktoren wie Ordnung und verständliche Anordnungen zu bewerten. Ein Mockup der Stationsseite ist in Abbildung x.y zu sehen.

Hier abbildung von mockup der Stationsübersicht einfügen

Dort wird bereits zu beginn auf die verschiedenen Subseiten geachtet. Diese sollen die Datenmenge in für die Nutzer besser verständliche kleinere Teile aufspalten und ordnen. Des weiteren gilt es zu beachten, dass durch die Struktur von Templates in Laravel eine einheitliche Ansicht für alle Stationen gegeben ist. Nur der Inhalt der Seiten unterscheidet sich von Station zu Station. Ein weiterer Vorteil ist die Nutzung von Bootstrap. DIeses Webframework nutzt CSS und Javascript, um je nach Webbrowser und Auflösung die Website trotzdem anschaulich darzustellen. So soll die Website auf dem Smartphone ohne spezielle App genauso gut benutzbar sein, wie auf dem heimischen Computer der Nutzer. Dabei ist die Ladedauer und die Größe der ausgelieferten Webseiten bereits beachtet. Die Größe ist immernoch von Relevanz, da die Nutzer noch mit geringen Bandbreiten auf Edge oder GPRS Geschwindigkeiten unterwegs sein können. In Tabelle x.y ist ein Vergleich der Ladezeit zwischen zwei Webseiten aufgezeigt.

Tabelle mit Ladezeit pro Website und Netz anlegen und füllen, eventuell erklärung was ist gprs und edge

6.3.1 Idee des dynamischen Nachladen

Hier was zum gedanken nicht alle statistiken direkt zu laden = langsam und viele daten + serverlast

Bei der erstellung der Übersichten und Statistiken für die Züge und die Stationen sollen immer nur die dem Nutzer sichtbaren Elemente erstellt und geladen werden. Dies spart Ressourcen auf dem Server und gleichzeitig Bandbreite beim Nutzer. Desweiteren ist die Webseite dadurch deutlich Performanter, da die Menge an QUellcode im Hintergrund besser aufgeteilt wird. Diese Unterteilung der Statistiken sorgt also nicht nur für eine bessere Übersichtlichkeit, sondern auch für einen schnelleren Seitenaufbau beim Nutzer. In Abbildung x.y wird das laden der Subressourcen aufgezeigt. Dieses erfolgt via Javascirpt code, welche die nicht sichtbaren element gleichzeitig im Hintergrund aus dem DOM entfernt.

Hier Abbildung einfügen mit nachladenden SUbseiten

6.3.2 Die Stationsübersicht

Auf der Seite der Stationsübersicht wird dem Nutzer eine Übersicht über die vorher ausgewählte Station angezeigt. Da es viele verschiedene Statistiken gibt, wird die Navigation durch Tabs realisiert. Die folgenden Elemente sind auf der Stationsübersicht wählbar:

Die Elemente anpassen sollte sich was ändern am Namen oder Inhalt. DER NAME/INHALT STIMMT NICHT

Fahrplan Sollte Fahrplan heiße

Gleisstatistiken Sollte Gleisstatistikeen werden. Hier gleiswechsel auswerten

Stundenstatistiken Wie oft fährt im Schnitt ein zugklasse x ab, wie oft fährt ein zug auf gleis x pro stunden

Tägliche Statistiken Hioer Verspätung und Gleiswechsel pro Tag

Haltestellenstatistik Hier was zur Haltestelle allgemein, wv zuge gesamt recorded und wv ausfall in Prozent, wv verspätungen Barschart wie damals ?

6.3.3 Die Zugübersicht

Auf der Seite der Zugübersicht wird dem Nutzer allerhand Informationen zu dem ausgewählten Zug angezeigt. Um die Informationen besser zu Ordnen wird eine Navigation mit Tabs erstellt, welche die folgenden Elemente enthält:

Die Elemente anpassen sollte sich was ändern am Namen oder Inhalt

Haltestellen Hier wird dem Nutzer die Route des Zuges angezeigt. Gleichzeitig gibt es einen Querverweis auf die angefahrenen Haltestellen, um deren Statistiken anzuschauen.

Verspätung Hier wird dem Nutzer eine Statistik zur Verspätung über den Verlauf der Strecke angezeigt. So sollen etwaige Engpässe aufgedeckt und erkennbar werden.

Ausfallstatistik Hier soll dem Nutzer eine Statistik zur Ausfallwahrscheinlichkeit angezeigt werden.

Gleiswechsel Hier kann der Nutzer sehen, ob der Zug in einem Bahnhof häufiger von einem anderen Gleis als dem Sollgleis abfährt.

Streckenwechsel Hier kann der Nutzer die verschiedenen Strecken sehen, im Falle eine Umleitung in der Vergangenheit

Verlauf Hier werden dem Nutzer alte Daten angezeigt und diese Ausgewertet.

6.4 Testen der Seiten mit Unit Test

Tests sind immer wichtig um bei änderungen am code zu merken, ob was schief läuft.

Mit einer steigende Komplexität der Website wird das manuelle Testen von Hand immer Aufwändiger. Um bestehende Seiten auf Fehler durch eine Änderung schnell zu überprüfen werden Unit Tests eingesetzt. Neben den Unit Tests werden Integration Tests durchgeführt, um ein fehlerfreies Zusammenarbeiten der Komponenten als Gesamtes sicherzustellen. Als einfachste Testart lässt sich eine Überprüfung von HTTP Status Codes realisieren. SO kann geprüft werden, ob eine Route den Code 200 (OK) oder eine Fehlercode zurück gibt. Im Falle eines internen Fehlers in Laravel, wird dem Nutzer eine benutzerdefinierte Fehlerseite angezeigt. Diese wird mit dem HTTP Code 500 (Internal Server Error) an

den Nutzer gesendet. In der Entwicklungsumgebung werden Debug Informationen dem Entwickler ausgegeben. Diese werden im Produktiveinsatz aus Gründen der Sicherheit nicht an die Nutzer ausgegeben. In Abbildung x.y ist eine Fehlermeldung aus Entwicklersicht zu sehen. Der Fehler wird zuvor bereits von einem Integrationstest erkannt und in einer Logdatei mit weiteren Details vermerkt. Diese Logdatei kann durch das Ausführen des Testframeworks angezeigt werden. Die Ausgabe ist in Abbildung x.y zu sehen. Die Vorteile von automatischen Tests ist die schnelle Erkennung, ob eine Änderung im Quellcode ungewollte Effekte verursacht.

Hier weiter Schreiben.....

Abbildung eines Fehlers auf der Website

Abbildung eines Testdurchlaufs Asser 5 Success 4 Error 1 oder so

6.5 Visualisierung der Datensätze

Dimensionen der Daten, ORT; ZEIT; NAME; STRECKE; etc.

Zur Visualisierung der Datensätze werden vorberechnete und LiveDaten verwendet. Je nachdem, ob dem nutzer eine interaktionasmöglichkeit gegeben werden soll, wird dies entschieden. Die dafür verwendeten tools stammen aus dem Python Modul matplot oder aus der javascript Libary d3.js.

Verlinken auf mathplot und d3js

Die Anzahl der Dimensionen der Datensätzen macht es schwierig zu entscheiden, ob eine Ansicht oder Grafik für diese Überhaupt Sinn ergibt. Daher werden zu Beginn der Visualisierungsprozesses verschiedene Techniken ausprobiert. Alle Skripts werden mit dem Bedacht auf die Wiederverwendbarkeit geschrieben. Ein wichtiger Schritte von der Datenbank zur fertigen Grafik ist die SQL-Abfrage. Diese soll optimiert sein, um den Datenbankserver nicht unnötig zu belasten. Dafür kann das MySQL Schlüsselwort EXPLAIN verwendet werden.

Verlinkung auf Explain bzw. erklären mit Screenshot

Nachdem die Abfrage ausgeführt ist, wird die Antwort als Objekt abgespeichert, hier gibt es grundlegende Unterschiede, ob das Objekt weitere Methoden enthält, oder ob das Objekt als simple Datensammlung darstellt.

Um die Datenmenge für die Nutzer zu verringern, wird die Datenaufbreitung serverseitig durchgeführt. Der Nutzer bekommt daraufhin vorverarbeitete Datensätze, welche mit als simples Datenformat oder JSON in eine Grafik eingebettet werden. Die Formate unterscheiden sich abhängig von den anzuzeigenden Statistiken. Für die einfachere wiederverwendbarkeit wird ein Controller für diese Datenaufbereitung entwickelt. Dieser wird GraphController.php genannt und soll intern die Datenvorverarbeitung im webserver übernehmen. Je nachdem wie die Python Scripte aufgebaut sind, können diese entweder vom Nutzer per WebCGI oder durch die Backendschnittstelle des Laravel Frameworks ausgeführt werden. Ein Vorteil bei der ausführung durch laravel ist der Cache, welche in

diesem Fall problemlos mit anderen Nutzern geteilt werden kann, da keine persönlichen Nutzerdaten darin vorhanden sind.

Grafik von CGI zu User mit und ohne Cache.

Da die Website komplett dynamisch generiert wird, müssen alle Querverweise durch die in Laravel vorgesehenen Routen ersetzt werden. Dies sieht im ersten Moment etwas ungewohnt aus hat aber den Vorteil, dass beim ändern der realen Adresse die Datei nicht mehr bearbeitet werden muss, da die Verlinkung durch das Framework vorgenommen wird.

Grafik einer Verlinkung , deren dynamische ersetzung.

Für jede Haltestellen sollen verschiedene Statistiken den Nutzern zur verfügung gestellt werden. Hierzu ist es notwendig sich gedanken über die möglichen relevanten Themen zu machen. Eine interessante Betrachtung ist zum Beipsiel die Verteilung von verschiedenen Zugklassen (ICE, RB, ...) auf die im Bahnhof vorhanden Gleise. Am Beispiel Karlsruhe kann erkannt werden, dass die Gleise 101 und 102 nicht für den Fernverkehr verwendet werden. Gleichzeitig kann die Verteilung der Zugklassen pro Gleis relativ zueinander erkannt werden. So gibt es Gleise welche hauptsächlich vom Fernverkehr bedient werden und Gleise, welche häufig für S-Bahnen benutzt werden. Das sich daraus weiter Informationen gewinnen lassen ist deutlich beim Berliner Ostbahnhof zu erkennen. Dort fahren die S-bahnen auf den ausgebauten Gleisen, welche über eine Stromschiene verfügen. Züge in der Nacht wie der NightJet verkehren dabei hauptsächlich auf den Gleisen eins bis drei. Als besondere Herausforderung beim Programmieren der Anzeige der Statistik kann das noch relativ unbekannte Framework c3js gesehen werden. Die Datensätze aus der Datenbank müssen bevor sie an den Nutzer gesendet werden als JSON Formatiert werden. Diese Aufgabe übernimmt der GraphController des Backends. Dieser liefert für die verschiedenen Statistiken die jeweiligen Ausgaben als JSON. Eine Herausforderung kann die Begrenzung des PHP Memory Limts sein, da bei großen Datenabfragen dieses leicht überschritten werden kann. Weitere Probleme treten in Verbindung mit Offset Bugs auf, diese sind durch die von extern kommende Programmteile vorprogrammiert. Oftmals ist ein Index eines Gleises nicht sichtbar, da die Ausführung der Javascript Funktion einen Index zu früh aufhört und somit den letzten Datensatz verschluckt. Dieses Problem kann mithilfe von weiteren Datensatzes am Ende behoben werden, diese werden, da die Zugklasse auf NONE gesetzt ist nicht im Frontend angezeigt. Dies Funktion des backends für die Gleisbelegunsstatistik wird in Quellcode Listing x.y dargestellt. Die darin verwendete MySQL Abfrage ist relativ unbelastend für den Server, da dieser alle Einträge der Station durch eine vorherige Query bereits zwischengespeichert hat und nur eine neue Aggregatfunktion über diesen Zwischenspeicher laufen lassen. Der auf die Abfrage folgende Quellcode sorgt für eine Formattierung der Datensätze in einem für c3js günstigen Ausgangsformat. Die generierte JSON Datei ist Exemplarisch in Listing x.y abgebildet.

Um die Daten in der JSON nicht dauerhaft erneut generieren zu müssen, wird der in Laravel bereits integrierte Cache benutzt. Dieser ist sehr mächtig und verfügt über verschiedene Routinen, welche verschiedene Speichermethoden des Caches unterstützten. Entschieden wurde sich für einen Dateibasierten Cache ohne weitere Software. Dieser ist in der Regel ausreichen schnell und wird vom Webserver im Arbeitsspeicher gehalten.

Die Ladezeiten einer Station nachdem diese im Cache vorhanden ist, fällt von über 250 Millisekunden auf unter fünf Millisekunden. Dies entspricht dem Faktor 50. Gerade bei den daten der Stationen ist ein Cache ohne größere Probleme umsetzbar. Da der Miner jede Station maximal einmal die Stunde aufruft, werden dem Nutzer auch keine Daten längerfristig vorenthalten. Zudem verändern sich die bereits gespeicherten Datensätze nicht mehr. Ein Nachteil des Caches ist bei der Entwicklung ebenfalls nicht zu merken, da hier die Konfigurationsdatei auf geringer oder gar keine Cachenutzung global eingestellt werden kann.

Listing von der Funktion GraphController@getTrainclassPerPlatformStatistic

Listing von der Susgabe JSON GraphController@getTrainclassPerPlatformStatistic

Hier Grafik von Gleisbelegung Karlsruhe HBF einfügen

Eine weitere Interessante Statistik könnte die Verspätung eines Zuges an verschiedenen Tagen sein. So kommt ein Zug zum Beipsiel chronisch zu spät oder es gibt häufig ungewollte Gleiswechsel eines Zuges. Vor allem Muster sollten am Ende von den Nutzern erkannt werden können. Die Beziehungen und Muster von Zügen untereinander ist auch bei einer Vorhersage mithilfe eines neuronalen Netzes wichtig. Diese Muster erstmals zu erkennen ist die Grundlage für eine spätere Aufbereitung der Daten für das neuronale Netz. In der Theorie müsste das neuronale Netz diese Muster selbständig erkennen und erlernen können, dies dauert aber einige Zeit und benötigt viele Datensätze, daher soll vorab eine Sortierung der Datensätze vorgenommen werden.

Da der Nutzer eine hübsche Statistik sehen will, werden verschiedenen Grafiken und Diagramme je nach Anwendungsbereich benutzt. Eine Idee für ein Diagramm wäre ein dreidimensionales Histogramm über die Zeit mit dem Streckenverlauf und der Verspätung. Ein solches Histogramm wird derzeit im Tensorflow eigenen Tensorboard verwendet, um die Verteilungen der einzelnen Schichten im Netzwerk über die Lerndauer zu visualisieren.

Histogramm aus tensorboard anzeigen

6.6 Darstellung der Datensätze

Hier was zum View Stations schreiben, un deren details

In der Bahnhöfsübersicht der Website werden den Nutzern alle Informationen und Statistiken zu diesem Bahnhof angezeigt. In der Fahrplanübersicht zu jedem Tag soll der Nutzer die Möglichkeit erhalten, zu jedem Zug Statistiken zu dessen Verspätungen über einen Zeitraum anzeigen zu lassen. Die AUswahl des Zuges findet entweder über den Fahrplan auf der Seite oder über die eigenständige Seite für die Zugsuche statt. Die Statistiken für jeden Zug werden, sofern nicht vorhanden automatische generiert und dann für 60 Minuten gecached. Der Einsatz eines Caches ist sinnvoll, da die Generierung der Statistiken Einige Ressourcen benötigt und sich die Datensätze eines Zuges nur selten ändern. Somit wird das doppelte Senden von Anfragen an den MySQL Server verhindert. Die Routen der Website sind darauf ausgelegt möglichst kleine Teile der

Website auszutauschen oder dynamisch nachzuladen. Dies soll vor allem bei mobilem Nutzen der Website die Ladezeiten gering halten und den Server entlasten.

Grafik Fahrplan bzw. Suche Zug mit zugnummerfull

Erstes Laden der Seite, Weiteres laden der seite (wie tcp syn,act,etc. DIagramm)

Kapitel 7

Schlussfolgerung

7.1 Rückblick

Was ist geschehen, was würden wir anders machen, was waren wichtige Schritte

7.2 Fazit

Ergebnis der Studienarbeit, was war gut, was war schlecht, hat alles so geklappt, wo gab es Probleme, wie wurden diese gelöst (kurz und knapp zusammengefasst.

7.3 Ausblick

Wie geht es weiter, könnte es weiter gehen, was sollte verbessert werden, wo befinden sich Schwachstellen, event. ungelöste Probleme

Weitere arbeit an dem Model+ vorhersage, weitere visualisierungen, bessere nutzerinteraktion

Liste der ToDo's