Challenges of Big Data, Artificial Intelligence and Vehicle Data

April 2022
Christian Prehofer
Presenter: Dr. Christian Prehofer

- 15+ years industry experience in telecom, Internet and automotive
  - Currently working at hdmi is

- 10+ years experience in university and applied research labs
  - Lecturer at TU München, supervised multiple PhDs
  - One successful startup on indoor positioning

- About 150 publications, 2 books / monographs
  - 7000 citations according to google scholar

- Recent focus on connected vehicles and Big Data Applications, AI
Big Data for Vehicle Data Analysis

- **Big Data for connected vehicle applications**
  - Vehicles generate enormous amount of data
  - Where to process? **In-vehicle, edge and cloud**

- **Use case driving behavior & energy efficiency**
  - Compute efficiency for every second
  - **Comparison** of Big Data processing options

- **Use case driver status monitoring**
  - Privacy preserving data analysis with federated learning

- **Discussion and Outlook**
Motivation – Big Data and Data Analysis in Automotive

Vehicle Driving Data Applications:
- e.g. insurance, eco driving,
  predictive maintenance,
  ADAS / AD optimization,…

2TB/day from internal CAN bus

Vehicle Sensor Data:
2 TB/hour
Applications of Connected Vehicles

- **Enhancing in-vehicle functions**
  - Routing and traffic data
  - Energy efficient driving
  - Enhanced autonomous driving functions

- **New services**
  - Insurance based on actual driving
  - Car sharing
  - In-car payment (fuel, ...)

- **Management**
  - Predictive maintenance
  - SW / function updates
IoT – Edge – Cloud Overview

**Vehicles & IoT**
- Lots of sensors & actuators
- Limited computing, cooling needed

**Edge Computing:**
- Local computing
- Faster response possible (wrt cloud)
- Local data only

**Cloud computing**
- Fully managed HW / SW, reliability, security
- Economies of scale reg. HW/energy/maintenance/utilization
Big Data for Vehicle Data Analysis

- **Big Data for connected vehicle applications**
  - Vehicles generate enormous amount of data
  - Where to process? **In-vehicle, edge and cloud**

- **Use case driving behavior & energy efficiency**
  - Compute efficiency for every second
  - **Comparison** of Big Data processing options

- **Use case driver status monitoring**
  - Privacy preserving data analysis with federated learning

- **Discussion and Outlook**
Example: Harsh breaking

- Find out breaking phases based on speed and acceleration
- Hard brake: deceleration is greater than a certain threshold ....
Use Case: Energy Efficiency

• Public **data set** (>500 trips, 8000km), incl.
  • Location
  • Speed
  • Energy consumption
  • Air conditioning, heating
  • Vehicle information (weight), ...

• Calculate „**needed energy**“
  • **VSP**: Vehicle specific power
  • Need road inclination (from GPS coordinates), acceleration etc

---

https://github.com/gsoh/VED/blob/master/README.md
Use Case: Comparing Used and Needed Energy

Driving data for e-vehicles
- Speed
- Uphill/downhill
- Vehicle weight

Calculate needed energy: VSP (Vehicle Specific Power)

\[ VSP \left[ \text{W} \right] = \frac{\text{Power}}{\text{Mass}} = \frac{\frac{d}{dt}(E_{\text{kinetic}} + E_{\text{potential}})}{m} + F_{\text{rolling}} \times v - F_{\text{aerodynamic}} \times v \]

Simplified version

\[ VSP \approx v \cdot [a \cdot 1.1 + 9.81 \cdot \text{grade} + 0.213] + 0.000426 \cdot v^3 \]

Vehicle energy consumption
- KWh from e-vehicle data
- Consider AC and heating
- Temperature, Battery SOC

Compare needed vs actual energy
- Energy efficiency calculation
  - different driving phases
- Energy in different temperatures
Use Case: Energy Efficiency Analysis

Data Set from E-Vehicles
>500 trips, 8000km

Needed vs used Energy
- Calculate physically energy needed for movement, „VSP“
- Compare VSP to actual power consumption, for every second

- Evaluation with Apache Spark, batch processing
Example in more detail: VSP vs Actual Power

More Details:
E-Vehicle Data with Uphill/Downhill

<table>
<thead>
<tr>
<th>veh_trip_id</th>
<th>total_time</th>
<th>total_power_kW</th>
<th>total_VSP</th>
<th>Correlation(Power,VSP)</th>
<th>max_Temp °C</th>
<th>min_Temp °C</th>
<th>total_airConPower_kW</th>
<th>total_heaterPower_kW</th>
<th>POW:VSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>455_1720</td>
<td>1719.0</td>
<td>10470.095412500792</td>
<td>9865.72035081531</td>
<td>0.948127956012671</td>
<td>15.0</td>
<td>13.5</td>
<td>0.0</td>
<td>0.0</td>
<td>1.0618637600148808</td>
</tr>
</tbody>
</table>

**Issues with altitude due to slow GPS receiver**

*Note: Altitude is computed from GPS coordinates by external website*
E-Vehicle Energy Consumption wrt Temperature

- Compute ratio between power and VSP for each a full trip
- Aggregation of 370 trips into temperature bins, total 4731 miles
- Clearly shows efficiency loss for colder temperatures
Data Stream processing (for same use case)

Apache Flink Stream processing

- Apache Flink as the true streaming processing engine
- The core of Flink is streams and transformations on dataflows
  - Many APIs, incl DataStream and SQL

- Note: Apache Flink mainly designed for online stream processing, Spark for batch.
  - Spark can do stream processing (with micro-batches), Flink can do batch
Data Stream processing

Apache Spark Big Data processing

- The core of Spark is generalized “map reduce”
- Functions operating on Dataframes, highly parallel
- High-level APIs, incl SQL and window operations

<table>
<thead>
<tr>
<th></th>
<th>Spark</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guarantee</td>
<td>Exactly once</td>
<td>Exactly once</td>
</tr>
<tr>
<td>Latency</td>
<td>High</td>
<td>Very Low</td>
</tr>
<tr>
<td>Throughput</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Computation model</td>
<td>Micro-batch</td>
<td>Streaming</td>
</tr>
<tr>
<td>Overhead of fault tolerance</td>
<td>Low</td>
<td>Low</td>
</tr>
</tbody>
</table>

Apache Spark Scalability on Similar Use Case

- Implemented with Spark SQL APIs
- Extensive use of window operations
- Stand alone mode
- Graph shows computing time in s

**More details:**

<table>
<thead>
<tr>
<th>Machine</th>
<th>Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Machine A</td>
<td>190</td>
</tr>
<tr>
<td>Machine B</td>
<td>22</td>
</tr>
</tbody>
</table>

Intel i3, 2 Core, 1.7 Ghz laptop  
Intel XeonW, 8 Core 3.7GHz, server
Apache Flink Performance on Use Case

- Using Flink DataStream API
  - Expressive window operations
  - Stand alone mode

![Graph showing performance comparison between two machines](image-url)

**Machine A**
- Intel i3, 2 Core, 1.7 Ghz laptop
- Number of Vehicles: 15000

**Machine B**
- Intel XeonW, 8 Core 3.7GHz, server
- Number of Vehicles: 45000
Spark and Flink Throughput for Vehicle Data Use Case

<table>
<thead>
<tr>
<th>Throughput for Vehicle Data Use Case</th>
<th>Spark</th>
<th>Flink</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel i3, 2 Core 1.7GHz</td>
<td>0.35MB/s</td>
<td>3–4MB/s</td>
</tr>
<tr>
<td>Intel XeonW, 8 Core 3.7GHz</td>
<td>3 MB/s</td>
<td>9–12MB/s</td>
</tr>
</tbody>
</table>

Notes:
- **Performance** can depend significantly on use case and **selected APIs**
  - Flink has powerful window operations in DataStream APIs
  - For Spark, we had to use SQL APIs
  - Includes complete execution, incl startup
Data Stream Processing with Apache Flink

- Apache Flink as the streaming processing engine
- The core of Flink is streams and transformations on dataflows
  - Many APIs, incl DataStream and SQL
## Performance Comparisons: Vehicle data streams

<table>
<thead>
<tr>
<th>Workstation, Intel Xeon W 3.7GHz, 8 Core, 3000 Euro</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Vehicle data streams</td>
<td>45k</td>
<td>60k</td>
</tr>
<tr>
<td>Average Latency range (ms)</td>
<td>1000 to 1800</td>
<td>1000 to 1800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Raspberry Pi 4b, ARM 7, 1.5 GHz, 4 Cores, 70 Euro</th>
<th>Scenario 1</th>
<th>Scenario 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Vehicles</td>
<td>12k</td>
<td>8k</td>
</tr>
<tr>
<td>Average Latency range (ms)</td>
<td>1000 to 2500</td>
<td>1000 to 3000</td>
</tr>
</tbody>
</table>
**Distributed Processing with Apache Flink - Distributed**

**Scenario 2, distributed with 2 devices:** Using two Flink engines/clusters

1. **Data Collection (e.g. Apache Kafka)**
2. **Data Preparation**
3. **Driving Behavior & Power Efficiency**

- **Flink cluster, Kafka Broker & Zookeeper**
- **Flink cluster, Kafka client**

© DENSO CORPORATION. All Rights Reserved.
## Evaluation: Performance on Distributed System

<table>
<thead>
<tr>
<th>Scenario 2, Distributed</th>
<th>Intel NUC Core i5, 2 Core 2.2GHz</th>
<th>Raspberry Pi 4b, ARM 7, 4 Cores 1.5 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum number of vehicle data streams</td>
<td>30k</td>
<td>30k</td>
</tr>
<tr>
<td>Throughput</td>
<td>6MB/s -9MB/s</td>
<td>6MB/s -9MB/s</td>
</tr>
<tr>
<td>Average Latency range (ms)</td>
<td>1000 to 1300</td>
<td>0 to 1200</td>
</tr>
<tr>
<td>CPU Utilization %</td>
<td>40% to 70%</td>
<td>60% to 90%</td>
</tr>
</tbody>
</table>

Promising results for 2 Pi 4b’s!
Big Data for Vehicle Data Analysis

• **Big Data for connected vehicle applications**
  • Vehicles generate enormous amount of data
  • Where to process? **In-vehicle, edge and cloud**

• **Use case driving behavior & energy efficiency**
  • Compute efficiency for every second
  • **Comparison** of Big Data processing options

• **Use case driver status monitoring**
  • Privacy preserving data analysis with federated learning

• **Discussion and Outlook**
1. Learning with local data in cars to create local model
2. Models are merged from different vehicles/drivers (no image data upload!)
   1. Exchange only NN parameters
3. Improves privacy + data volume
Input Data: NTHU Dataset

- 36 people of different genders and ethnicities
- Total 9 and a half hours (varying length videos)
- Annotated per frame (Eye, Mouth, Head, Drowsiness)
- Train, Val, Test Split (after preprocessing):

<table>
<thead>
<tr>
<th></th>
<th>Training</th>
<th>Validation</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Subject</td>
<td>18</td>
<td>4</td>
<td>14</td>
</tr>
<tr>
<td>Number of videos</td>
<td>288</td>
<td>16</td>
<td>56</td>
</tr>
<tr>
<td>Number of annotations (per-frame)</td>
<td>537,245</td>
<td>145,049</td>
<td>596,590</td>
</tr>
</tbody>
</table>

[Weng et al., “Driver Drowsiness Detection via a Hierarchical Temporal Deep Belief Network.”]
Federated Learning for Driver Drowsiness detection

- **Use case of driver drowsiness detection**
- Detection Method:
  - **Using Driver’s Behavioral measurements (eyes, mouth, head etc.)** non-intrusive
    - Datasets used: **NTHU DDD**, others, e.g. DROZY, UTA-RLDD
    - **Features**
      - **PERCLOS**: PERcentage of eye CLOSure
      - **FOM**: Frequency of Mouth Open

Further Details:
FL: Our Setup

- Hyperparameter Selection
  - K: Total number of clients used in the process = 18
  - C: Fraction of clients used at each iteration = 2
- Data Sampling

Local dataset at each round: 4 video sequences for varying lengths for a set based on external factors. e.g. night time videos where the driver is wearing glasses.

The 4 videos have different facial actions.
Baseline Model Architecture

Stage 1: Multi-task CNN

- **X:** Images as Arrays
- **y:** eye_label, mouth_label

Stage 2: Drowsiness Detection from Behavioral Parameters

- Calculate PERCLOS, FOM
- PERCLOS
  - **y:** drowsy/not
- FOM
  - **y:** drowsy/not
- Drowsiness Detection
  - metrics

**PERCLOS** = \( \frac{N_{\text{close}}}{N_{\text{total}}} \times 100 \)

**FOM** = \( \frac{N_{\text{open}}}{N_{\text{total}}} \times 100 \)

Frequency range: All frames in video
Threshold values:
- PERCLOS bounded at 20%
- FOM bounded at 16%

[Savas et al., “Real Time Driver Fatigue Detection System Based on Multi-Task ConNN.”, Zhuang et al. “Driver Fatigue Detection Method Based on Eye States With Pupil and Iris Segmentation”]
Baseline Model Architecture: Stage 1

Cross-Entropy Loss

\[- \frac{1}{N} \sum_{n=1}^{N} \left[ y_n \log \hat{y}_n + (1 - y_n) \log(1 - \hat{y}_n) \right]\]

\(CE_{\text{eye}}:\) Binary Cross-Entropy Loss  
\(CE_{\text{mouth}}:\) Sparse Categorical Cross-Entropy Loss  

Total Loss:

\[L(y^\wedge, y) = CE_{\text{eye}} + CE_{\text{mouth}}\]

Accuracy:

\[\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}\]
Baseline Model Arch: Loss

Experiment: initial learning rate: $1 \times 10^{-2}$ (0.01), lr decay: 0.001, momentum: 0.99, batchsize: 64, epochs: 20, batchNorm on conv layers + dropout rate (20%) on fc layers
Baseline Model Arch: Final

Experiment: initial learning rate: $1 \times 10^{-2}$ (0.01), lr decay: 0.001, momentum: 0.99, batchsize: 64, epochs: 20, batchNorm on conv layers + dropout rate (20%) on fc layers

~ 65 %
FL: FedAvg and DynAvg

Training loss from our experiment ($\Delta = 0.5$) shows no improvement with non-IID data

1. Federated Averaging: averaging of all parameters
2. Dynamic Averaging: significant parameter changes updated only

Challenges with highly non-IID data set!

[Kamp et al., “Efficient Decentralized Deep Learning by Dynamic Model Averaging”]
FL: how much to aggregate from local updates

Training loss from our experiment (μ = 0.01) shows improvement

Promising Approach: FedProx
Regularization of loss function in local objectives to control divergence
Permit some local deviation from central model

[Li et al., “Federated Optimization in Heterogeneous Networks”]
Results: Comparing baseline vs FedProx

Predictive performance

New results with >80% accuracy in our labs. Needed more and high-quality data

Test Accuracy for baseline model (65%) and federated model (62%) for Eye Class
Big Data for Vehicle Data Analysis

- **Big Data for connected vehicle applications**
  - Vehicles generate enormous amount of data
  - Where to process? **In-vehicle, edge and cloud**

- **Use case driving behavior & energy efficiency**
  - Compute efficiency for every second
  - **Comparison** of Big Data processing options

- **Use case driver status monitoring**
  - Privacy preserving data analysis with federated learning

- **Discussion and Outlook**
Challenges in Data Science, based on 2017 Kaggle Survey

- Dirty data
- Lack of data science talent in the organization
- Company politics / Lack of management/financial support for a data science team
- The lack of a clear question to be answering or a clear direction to go in with the available data
- Unavailability of/difficult access to data
- Data Science results not used by business decision makers
- Explaining data science to others
- Privacy issues
- Lack of significant domain expert input
- Organization is small and cannot afford a data science team
- Team using multiple ad-hoc development environments such as Python/R/Java/etc.
- Limitations of tools
- Need to coordinate with IT
- Maintaining responsible expectations about the potential impact of data science projects
- Inability to integrate findings into organization's decision-making process
- Lack of funds to buy useful datasets from external sources
- Difficulties in deployment/scoring
- Scaling data science solution up to full database
- Limitations in the state of the art in machine learning
- Did not instrument data useful for scientific analysis and decision-making

https://www.kaggle.com/ash316/novice-to-grandmaster
### Big Data and Vehicle Data Analysis

Applications for vehicle data with **different requirements**
- Need to understand what data is needed, as well as timing requirements
- Computing in car, in cloud or in the edge

Compared the **architecture options in vehicular** systems,
- Trade-off computing power vs networking vs energy
- Need to consider application development and operation

**Performance and scalability of Big Data solutions**
- Apache Flink **scales down** to small machines (4 cores)
- **Distributed Big Data** processing can be highly efficient

**Privacy-aware distributed AI with federated learning**
- Promising first result on FedProx, currently ongoing work

---

Thanks to Shafqat Mehmood, Atiqa Zafar, Shumail Mohyuddin, Chih-Hong Chen, William Lindskog
Vehicle Energy Dataset (VED)

- About 8000 km of driving data with e-vehicles
- Detailed data with 1s sample time for speed, energy

<p>| TABLE II |
| CONTENTS OF TIME-STAMPED DYNAMIC DATA |</p>
<table>
<thead>
<tr>
<th>Data Name</th>
<th>Populated %</th>
<th>Sampling Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPS</td>
<td>100 %</td>
<td>3 sec</td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td>Vehicle Speed (km/h)</td>
<td>100 %</td>
<td>1 sec</td>
</tr>
<tr>
<td>Engine RPM (rev/min)</td>
<td>99.98 %</td>
<td>2 sec</td>
</tr>
<tr>
<td>Mass Air Flow (g/s)</td>
<td>100 %</td>
<td>2 sec</td>
</tr>
<tr>
<td>Fuel Rate (L/h)</td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td>Absolute Load (%)</td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td>Fuel Info</td>
<td>80 %</td>
<td>2 sec</td>
</tr>
<tr>
<td>Short Term Fuel Trim B1 (%)</td>
<td>89.2 %</td>
<td>5 sec</td>
</tr>
<tr>
<td>Short Term Fuel Trim B2 (%)</td>
<td>33.0 %</td>
<td>5 sec</td>
</tr>
<tr>
<td>Long Term Fuel Trim B1 (%)</td>
<td>79.8 %</td>
<td>30 sec</td>
</tr>
<tr>
<td>Long Term Fuel Trim B2 (%)</td>
<td>23.8 %</td>
<td>30 sec</td>
</tr>
<tr>
<td>Engine Info</td>
<td>0 %</td>
<td>5 sec</td>
</tr>
<tr>
<td>AirCon Power (KW)</td>
<td>0 %</td>
<td>60 sec</td>
</tr>
<tr>
<td>AirCon Power (W)</td>
<td>0 %</td>
<td></td>
</tr>
<tr>
<td>Heater Power (W)</td>
<td>0 %</td>
<td></td>
</tr>
<tr>
<td>Auxiliary Info</td>
<td>100 %</td>
<td></td>
</tr>
<tr>
<td>Battery SOC (%)</td>
<td>0 %</td>
<td>60 sec</td>
</tr>
<tr>
<td>Battery Voltage (V)</td>
<td>0 %</td>
<td>5 sec</td>
</tr>
<tr>
<td>Battery Current (A)</td>
<td>0 %</td>
<td>1 sec</td>
</tr>
<tr>
<td>Battery Info</td>
<td>100 %</td>
<td></td>
</tr>
</tbody>
</table>