Unsupervised Anomaly Detection using *H2O.ai*

Peter Schrott
Berlin Institute of Technology
peter.schrott@campus.tu-berlin.de

Julian Voelkel
Berlin Institute of Technology
voelkel@campus.tu-berlin.de

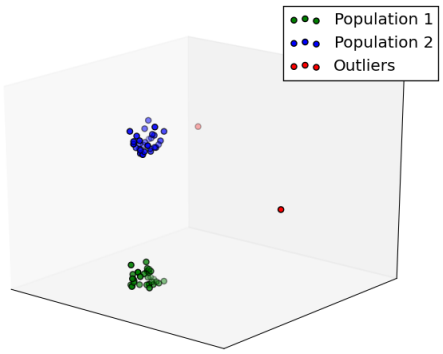


Figure 1: In this simple example, we can see two outliers within one dataset with two populations.

1. INTRODUCTION

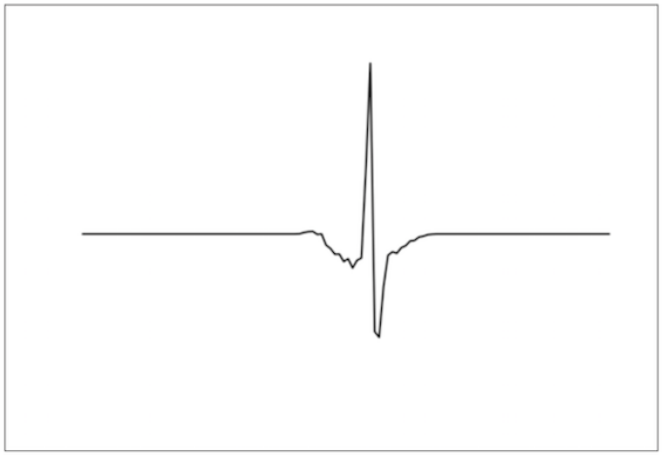
Anomaly detection (commonly referred to as *outlier detection*) is one of a few very common tasks in the field of Machine Learning. The goal of algorithms designed for the purpose of anomaly detection are concerned with finding data in a dataset that does not conform to a pattern. That means, the goal is to identify data points, that are special in regards to their behavior, compared to the data points in the dataset, that are considered "normal". [**?**] These data points are called outliers, because they show different behavior than one would expect. Figure 1 shows a basic plot containing data points, with two different populations, along with two outliers, that do not seem to fit either pattern.

The value of identifying outliers in a dataset lies in the action one can take after detecting them. Common applications of anomaly detection algorithms include among others health care and fraud detection. In the former, those algorithms can for instance help identifying sick patients, by identifying anomalous vital signs compared in a group of similar patients. In the latter application, those algorithms can account for fast, actionable information in case of credit card fraud, which can be identified by anomalous purchases, given the owners purchase pattern in form of historical purchases.

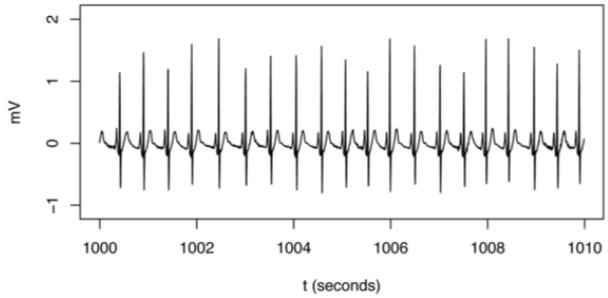
Anomaly detection can happen in a supervised, semi-supervised, as well as in an unsupervised fashion. The credit card fraud detection would be a semi-supervised learning task, since we can assume, that a new credit card will not be the subject of fraud for at least the first couple of purchases. Hence, we obtain a training set of "normal" purchases (i.e. data point) and can for each newly generated data point decide, whether it conforms to the pattern or does not. Since our project is focused exclusively on the unsupervised case where we do not know which data points are considered normal, but rather have to find a structure or pattern in the data first, in order to then be able to identify data points not conforming to the pattern, the next section will focus on unsupervised anomaly detection and the challenges we face in its context.

1.1 Challenges of Unsupervised Anomaly Detection

One difficulty that arises in unsupervised anomaly detection is, since we do not have any labels for training data, we do not even know what we are looking for. That is, we do not know what a "normal" data point would look like, let alone what an anomalous point would look like. To put it in Ted Dunning's and Ellen Friedman's words: "Anomaly detection is about finding what you don't know to look for." [**?**] Since there is no labeled training data in the most widely applicable case of unsupervised anomaly detection, the approach of finding outliers is a different one compared to training a model and then predicting to which class an unseen data point belongs to (much like binary classification). Instead, in case of unsupervised anomaly detection, we generally assume that the number of "normal" data points exceeds the number of anomalous data points by far.[**?**] This assumption is fundamental to unsupervised outlier detection, since we would not be able to learn what is normal otherwise, as a relatively large number of anomalous points would change the skew the structure of the data in a way, that would make determining what is "normal" impossible . Not being able to determine what "normal" is, means there is no way of finding what is anomalous. Since the goal of anomaly detection is finding what is anomalous, unsupervised anomaly detection usually starts with figuring out what "normal" is.



(a) Obvious outlier in small sample size...



(b) ...do not have to be outliers in the context of the whole dataset

Figure 2: Contextual anomaly

After achieving this (which, oftentimes, is much harder than it sounds), we can determine the deviation of a data point to what is "normal" using some similarity measure. There are, however, different algorithms dealing with the problem of unsupervised anomaly detection for different fields and problem domains. The main reason why there is no single approach applicable to each problem is, that there are tremendous differences in what is considered normal and what is considered anomalous, depending on the application domain we are looking at. Considering an example for this circumstance given in [**?**], one can easily imagine why that is the case: "The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus applying a technique developed in one domain to another is not straightforward." Besides the domain specificity of anomalies, there is also the context specificity of anomalies to keep in mind. This concept is illustrated in Figure 2, taken from [**?**]

Common techniques and algorithms used to perform unsupervised anomaly detection include clustering based methods, statistical techniques, information theoretic methods as well as spectral techniques. Note, however, that the choice of algorithm can depend heavily on the problem's domain.

1.2 Deep Learning Auto-Encoder

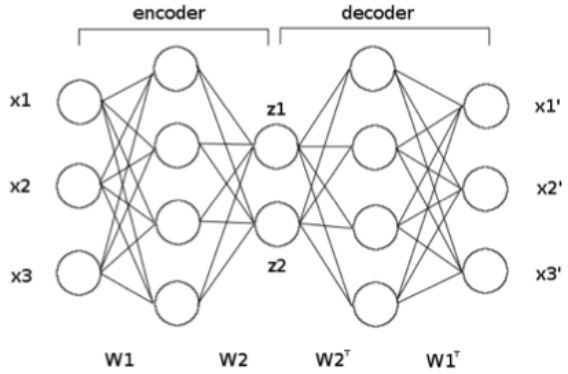


Figure 3: Schematic diagram of a basic auto-encoder with three input features

An auto-encoder, autoassociator or Diablo network is a specific type of artificial neural network. The goal of a deep learning auto-encoder is to learn a compressed encoding of a dataset. Due to that purpose, the auto-encoder consists of one input layer, one or more hidden layers and an output layer with equally as many neurons (i.e. features) as the input layer. In order to achieve the goal of representing a dataset in a compressed manner, the auto-encoder is given the original dataset as input, while the target output is the input itself. [**?**] The loss function is some type of dissimilarity function (typically a squared error function) between the input and the output of the auto-encoder. This way, the auto-encoder is forced to learn a nonlinear (or linear), compressed representation of the original dataset. This, of course, makes the auto-encoder a useful tool for dimensionality reduction. For the special case where there is only one linear hidden layer with k neurons and the mean squared error criterion is used to train the auto-encoder, the hidden layer consisting of the k neurons learns to represent the dataset in the dimension of its first k principal components. [**?**] This is much like Principal Component Analysis (PCA). If, however, the hidden layer is of nonlinear nature, then the auto-encoder behaves very different compared to PCA. [**?**]. Due to its ability to learn a compressed version of the dataset, the main application of the deep learning auto-encoder is obviously dimensionality reduction. In our case, though, we want to use the deep learning auto-encoder in order to perform unsupervised anomaly detection.
A schematic diagram of an auto-encoder taken from [**?**] is given in Figure 3.

2. PROBLEM STATEMENT

As already mentioned in **1. Introduction**, anomaly detection refers to the task of identifying observations, that do not match the general pattern of the data set they arise in. Oftentimes anomaly detection happens in an unsupervised context, which means that the dataset being operated on is unlabeled, and the goal is to identify exactly those samples, that fit the pattern of the dataset the least. This is also the case we want to investigate regarding the usability of a certain algorithm originally designed for a different purpose. Within the scope of this project, we analyze the performance of a particular algorithm more commonly used

in a field different to the one of unsupervised anomaly detection. Specifically, with this project, we aim at providing an answer or at least hints to the answer of the question: **Is a deep learning auto-encoder** (see section 1.2) **well suited for anomaly detection in an unlabeled dataset?**

2.1 Target

As stated above, target of this project is to evaluate the quality of a Deep Learning Auto-Encoder model for the task of identifying anomalies in an unsupervised context. Experiments comparing the performance of the deep learning auto-encoder with the performance of other algorithms in the same context shall indicate whether it is a good idea to use the auto-encoder in the context of unsupervised anomaly detection or not.

2.2 Scope

This project consists of various different steps in order to obtain an answer to the problem specified above. These steps can be outlined as follows:

1. Study an existing implementation of the Deep Learning Auto-Encoder model
2. Apply this implementation to a given unlabeled dataset
3. Compare the outcome to already existing outcomes of other algorithms
4. Draw conclusions about the general suitability of the algorithm based on the results produced by the application of its implementation compared to those of other algorithms

3. METHODOLOGY

Within the scope of this project, we use *H2O.ai*'s (see section 3.1) implementation of the deep learning auto-encoder model through its Sparkling Water API on top of Apache spark. The dataset we use in order to be able to compare our results to those of our peers using different algorithms is the *AXA Driver Telematics Analysis* dataset (see 3.2), which contains multiple vehicle traces by multiple drivers.

3.1 H2O.ai and H2O Deep Learning

H2O by H2O.ai is an open source software project primarily used for fast scalable machine learning. The software offers a predictive analytics platform, combining high performance parallel processing with an extensive machine learning library. [**?**] H2O was built on top of Apache Hadoop as well as Apache Spark. As of February 2015, the software has more than 12,000 users and is deployed more than 2,000 companies, including PayPal, Nielsen and Cisco. [**?**]
Besides Distributed Random Forests, K-means, Generalized Linear Model and Gradient Boosting Machine, H2O also offers readily available Deep Learning algorithms, which includes the auto-encoder.
As already mentioned in Section 1.2, the deep learning auto-encoder is an algorithm that is used primarily for dimensionality reduction. The conceptional way we want the auto-encoder to detect outliers in an unsupervised manner is shown in Figure 4. As already mentioned in section 1.2, the auto-encoder is forced to learn the identity of the data through a nonlinear, reduced representation of it. This is done by first reducing the data's dimensionality and then reconstructing it from that reduced representation. In 4a we see the original unlabeled dataset, where we want to identify outliers, without knowing what is normal. The auto-encoder reduces this dataset to some lower dimension - illustrated in 4b - where the reconstruction error of said reduced representation is lowest. Since one assumption in unsupervised anomaly detection is that the number of normal data points exceeds the number of anomalous ones by far (see Section 1.1), that learned model will be influenced more by what is normal in the data than by what is anomalous. Thus, attempting to reconstruct a data point from its reduced representation will have a greater error for anomalous data points that it will have for normal data points. This is illustrated in 4c, where data points belonging to either population are reconstructed with a small error, while clearly, the reconstructed outliers are far away from their respective original data point. Thus, the error (i.e. distance) between the original data point and its reconstruction is bigger than for "normal" data points belonging to one of the two populations.
Since the H2O deep learning auto-encoder returns confidence scores for each data point in the dataset, we have to choose a threshold, in order to decide which points we think are outliers and which are not.

x	y	DriverID
0	0	1
18.6	-11.1	1
36.1	-21.9	1
53.7	-32.6	1
.	.	.
.	.	.
.	.	.

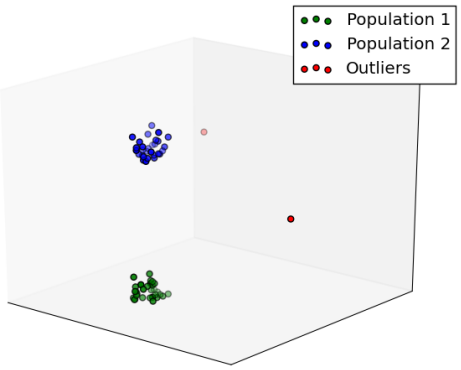
Table 1: The first few rows of the driver one's first drive

3.2 The AXA Driver Telematics Analysis Dataset

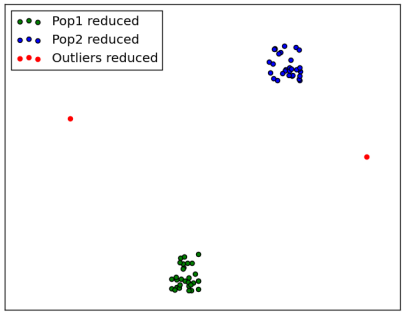
The AXA Driver Telematics dataset is a dataset that was being released to the public in form of a Kaggle challenge [1]
The dataset is a directory based dataset. This means meta data is implicitly contained in the directory and file structure of the dataset. In total there are logs for 2736 drivers. There is one designated folder for each driver, each of which contains 200 different drive traces in form of CSV files. In the raw data, a single drive is given by a single CSV file consisting of three columns and a varying number of rows. For every single drive, there is one column containing x coordinates one column containing y coordinates and one column containing the driver ID. Each row then represents the driver's position one second after the previous row. Every drive has been anonymized, such that each drive starts at position $(x, y) = (0, 0)$ and all the following coordinates have been randomly rotated.
For instance, the first few rows in the CSV file representing driver one's first trip are shown in Table 1

The catch with this dataset is that while there is a folder for each driver with a number of his or her respective traces, there is always a varying and unknown number of traces that

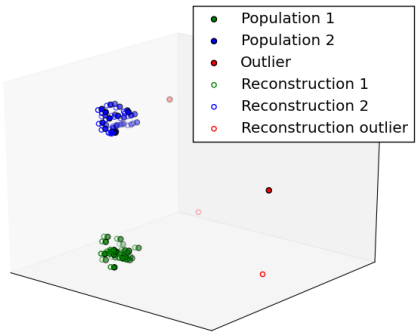
[1]Find the challenge with the dataset here: `https://www.kaggle.com/c/axa-driver-telematics-analysis/data`



(a) The given unlabeled dataset...



(b) ...is reduced in dimensionality...



(c) ...and reconstructed, in order to find outliers

Figure 4: Identifying outliers by reconstruction error

Figure 5: A visualization of the raw dataset. Each line is generated by connecting the (x, y) coordinates of one particular trip and is thus a visualization of that one particular trip

were being generated by other drivers (otherwise not represented in the dataset) in that particular folder as well. These unlabeled outliers is what we hope end up identifying using the deep learning auto-encoder.
Figure 5 (taken from kaggle.com) shows what the whole dataset might look like, if we just plotted each driving trip as a line connecting its consecutive (x, y) points.

With a size of 1.44 GB compressed and 5.92 GB in extracted state, this dataset can be considered reasonably large and thus, processing the dataset on a parallel system is justified.
The fact, that AXA, a major car insurance provider (amongst other things), releases a dataset of this kind with the goal of identifying anomalies in driving patterns to the public, hints at a real world application for anomaly detection algorithms that generates value of some kind. In this case, the companies goal might have been to identify or count the times a person not insured for a particular car still drove said car. Identifying those instances might for instance allow challenging of fraudulent insurance claims. Another way AXA might profit from identifying anomalies in drives is, that having an estimate of the numbers of uninsured drives allows for adjustment of internal calculations of revenue, deductions and the like, as well as adjustment of insurance rates. If not one of the above, there has to be some kind of incentive for AXA to be able to identify anomalies.
In our case, however, the fact that more than 1,500 teams already submitted their solutions, allows for some benchmarking of the deep learning auto-encoder.

3.3 Feature Extraction

In order to be able to find the anomalous driving trips for each driver contained in our dataset by applying the H2O Deep Learning auto-encoder, we have to extract features from our driving traces first. Ideally, those feature would

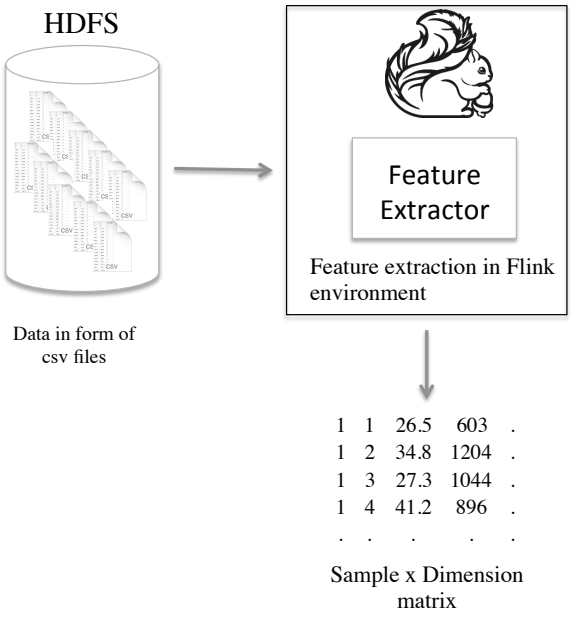


Figure 6: Schematic visualization of our feature extraction

be meaningful, with large variance in those driving trips that were generated by other drivers. We use Apache Flink in order to extract our features, as illustrated in Figure 6. Since every drive is initially given as a cvs file containing only (x, y) coordinates of that trip (see section 3.2), we implement a feature extraction engine that calculates different features for each driving trip. That includes for instance driven distance, time of the trip, average speed, average acceleration, et cetera. An overview of all the features we extract is given in the following list:

1. driverId
   The unique identifier of the driver this trip belongs to
2. driveId
   The among one driver unique identifier of a drive
3. duration
   Duration of the trip.
4. distance
   Distance driven in this trip.
5. speedMax
   Maximum speed for this trip.
6. speedMedian
   Median speed for this trip.
7. speedMean
8. speedMeanDeviation
9. speedSd
10. speedMeanDriver (*1)
11. speedSdDriver (*1)
12. accMax
13. accMedian
14. accMean
15. accMeanDeviation
16. accSd
17. accMeanDriver (*1)
18. accSdDriver (*1)
19. angleMedian
20. angleMean
21. turns35P (*2)
22. turns35N (*2)
23. turns70P (*2)
24. turns70N (*2)
25. turns160P (*2)
26. turns160N (*2)
27. turnsBiggerMean (*2)
28. turnsU (*2)
29. stopDriveRatio
30. stops1Sec
31. stops3Sec
32. stops10Sec
33. stops120Sec

After extracting all the features, we obtain one row of our The features we extract and ultimately pass to the autoencoder as input vectors, are of fundamental meaning for the quality of the anomaly detection. Depending on their significance for a driver's "fingerprint", the algorithm might perform well or it might produce unusable results in case the features do not bear a large significance for a driver's pattern of driving.

Figure 7 gives a small overview of the dataset by displaying the duration and distance of every single drive for a few drivers. What we can see here already, is a rather large variance in trip duration among different drivers. Also, there seem to be cases where the driver did barely move, as well as very short trips (short in the sense of time passed during the trip).

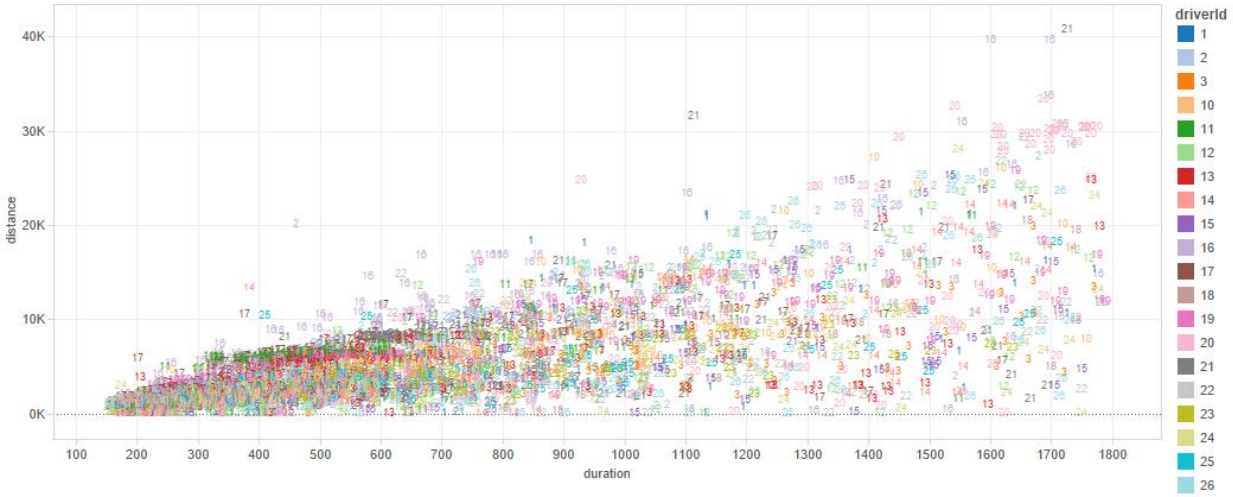


Figure 7: Plot of duration versus distance for each drive by each driver.

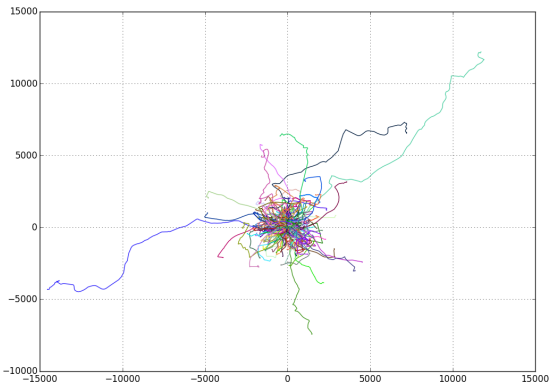


Figure 8: All drives of driver 18 visualized

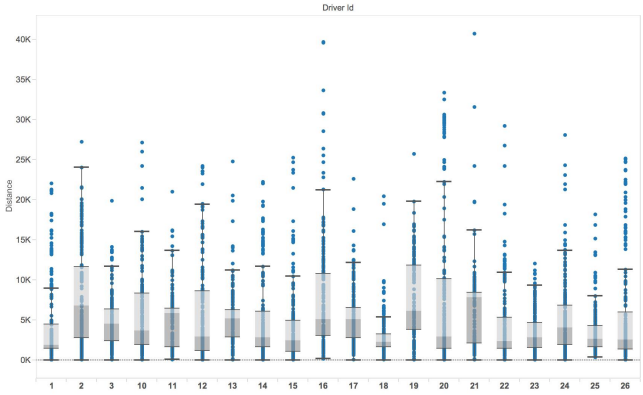


Figure 9: box-plot-distance

3.4 Exploratory Data Analysis

In order to get an overview of the data we are dealing with, we perform exploratory data analysis, mainly in the form of visualizations. Figure 8 provides some insights about driver 18. Since the plot contains all 200 recorded driving trips of that driver, including the drives not originating from driver 18, we can attempt to see some anomalies here. In this case, we can clearly see, only by looking at the driven distance, that a handful drives do not seem to fit the pattern of this particular driver., which seems to be to drive rather short distances.

Using the above mentioned features we extracted, exploratory data analysis yields some other insights about variation in driving pattern amongst drivers, as well as some anomalous behavior among driving trips for single drivers as well. The box plot in Figure 9 for instance shows not only the variation of driving distance among drivers, but also between different trips for one driver. In particular, this box plot seems to confirm the assumptions made about driver 18's driving pattern after having seen the plot of all that driver's trip in Figure 8. That is, because we can clearly see in the box plot , that driver 18 has a very low variation when it comes to trip length.

In Figure 10 we can see the drivers with their respective driving duration in seconds. One insight we gain from this plot is, that there seem to be quite a lot of traces with a very low recording time. Looking at the data points lying under the lower quartile bound, we can see, that for some drivers, the duration of the recorded drive is lower than 300 seconds in 25% of the cases (Driver 1, 18). We can also see, that some drivers tend to go on short drives in most cases. Drivers 11, 16 and 18 for instance seem to be going on drives with a duration of 10 or less minutes 75% of the time.

Considering the number of stops per driver and their respective driving trip is also helpful in detecting driving patterns. In Figure 11, size and color of the circle visualize the number of stops between 10 seconds and 120 seconds. The insights we gain from this plot can be the revealing of the area the respective driver commonly drives in. A constant high number of short breaks might indicate a driver mainly driving in urban areas. In this case, trips with more infrequent, shorter stops could be considered anomalous because the might originate from a rural driver.

4. EXPERIMENTS

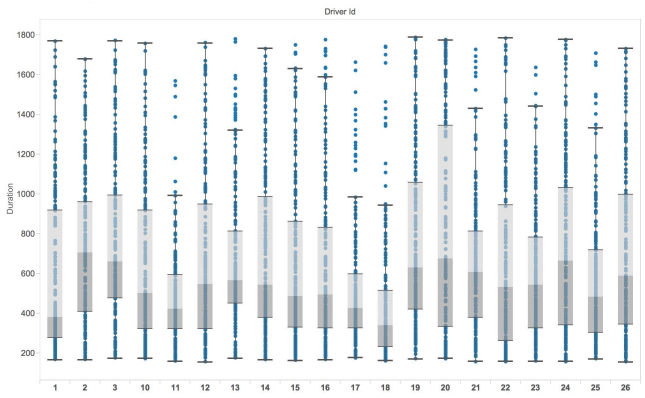


Figure 10: box-plot-duration

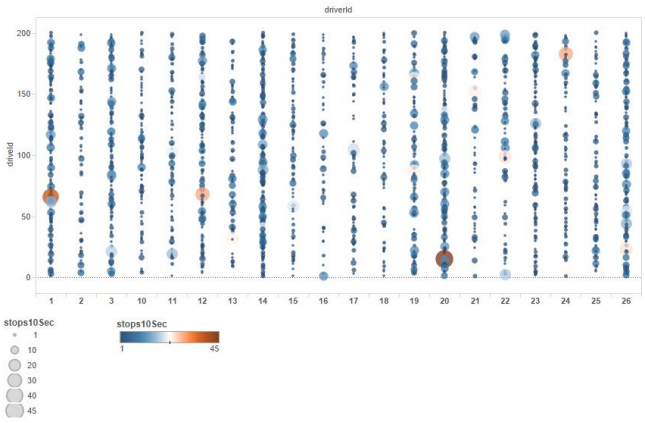


Figure 11: number-of-breaks-10-120

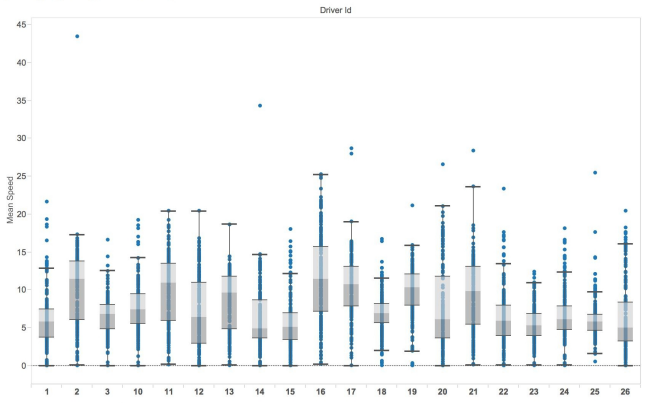


Figure 12: box-plot-mean-speed

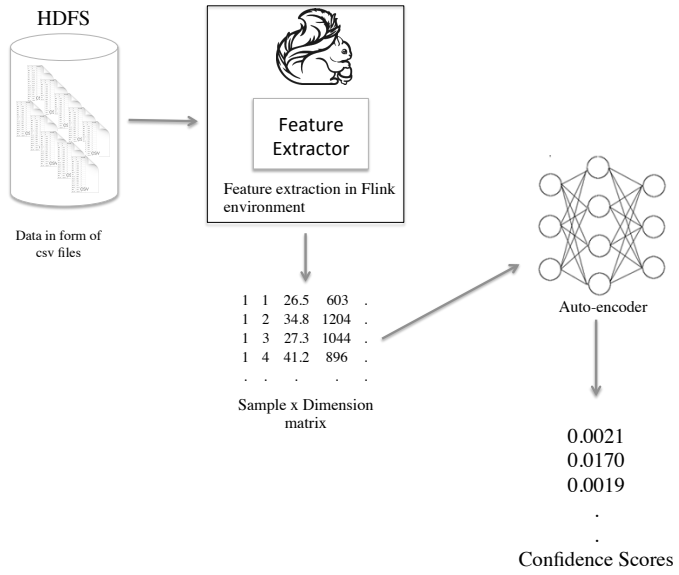


Figure 13: Schematic visualization of our experiment(s)

In order to be able to obtain indications as to whether the deep learning auto-encoder is suitable for anomaly detection, we have to somehow evaluate its findings. Since our dataset is of unlabeled nature, we do not have any labels. As stated in Section 3.2 though, the dataset was the subject of a Kaggle.com challenge. Hence, our prediction of outliers will automatically be graded after uploading it to the site, which will in turn give us the accuracy of our predictions. Figure **??** outlines the scope of our first experiment, that aims at getting indications as to whether a deep-learning auto-encoder is suitable for unsupervised anomaly detection.

5. RESULTS

6. CONCLUSION

I