An Evaluation Framework for Traffic Information Systems Based on Data Streams

Sandra Geisler[a,*], Christoph Quix[a], Stefan Schiffer[b], Matthias Jarke[a,c]

[a] *Information Systems, RWTH Aachen University, Ahornstr. 55, 52056 Aachen, Germany*
[b] *Knowledge-based Systems Group, RWTH Aachen University, Ahornstr. 55, 52056 Aachen, Germany*
[c] *Fraunhofer FIT, Schloss Birlinghoven, 53754 Sankt Augustin, Germany*

Abstract

Traffic information systems have to process and analyze huge amounts of data in real-time to effectively provide traffic information to road users. Progress in mobile communication technology with higher bandwidths and lower latencies enables the use of data provided by in-car sensors. Data stream management systems have been proposed to address the challenges of such applications which have to process a continuous data flow from various data sources in real-time. Data mining methods, adapted to data streams, can be used to analyze the data and to identify interesting patterns such as congestion or road hazards. Although several data stream mining methods have been proposed, an evaluation of such methods in the context of traffic applications is yet missing. In this paper, we present an evaluation framework for traffic information systems based on data streams. We apply a traffic simulation software to emulate the generation of traffic data by mobile probes. The framework is applied in two case studies, namely queue-end detection and traffic state estimation. The results show which parameters of the traffic information system significantly impact the accuracy of the predicted traffic information. This provides important findings for the design and implementation of traffic information systems using data from mobile probes.

Keywords: Data Streams, Car-to-X Communication, Data Stream Mining

1. Introduction

With increasing traffic volumes in cities, on highways and other main roads, the demand for accurate and real-time traffic information covering the complete road network is rising. Real-time traffic information can be utilized amongst others for real-time traffic management or for routing and travel time calculation in navigation systems. Commonly used services, such as the Traffic Message Control (TMC) system, rely partially on manually collected data and therefore, can only provide updates in intervals of several minutes. Furthermore, as updating the data requires also human effort, the information is often outdated. This prohibits real-time information being delivered to road users. Additionally, these services are most often restricted to motorways and major roads, and do not cover urban or rural streets.

*Corresponding author
Email address: geisler@dbis.rwth-aachen.de (Sandra Geisler)

In contrast, new possibilities in information technology enable efficient collection, processing, and distribution of traffic-related data. In addition to data from stationary sensors (e.g., inductive loops), data from mobile sources can be used in traffic applications and offers several advantages. As modern cars are equipped with various sensors and often also with mobile communication technology, data collected by the sensors in cars (called *Floating Car Data*, FCD) is available at a relatively low price, and it allows to cover the complete road network including rural roads and urban side roads. FCD can be send to a traffic management center for applications like traffic state estimation, but it can also be exchanged between a vehicle and the road infrastructure (Car-to-Infrastructure Communication, C2IC) or between two or more vehicles (Car-To-Car Communication, C2CC). This can either be done in a centralized fashion or by utilizing a peer-to-peer infrastructure as for example presented in [Delot et al. 2010]. Besides traffic state information, there is also a huge interest from industry and public authorities in traffic safety applications, such as intersection assistance systems, to reduce road fatalities. These applications, to be useful, of course require also very low processing and notification latencies. Therefore, the traffic information generation process, starting from the acquisition of raw traffic data, through data fusion and data analysis, up to the delivery of the derived and refined information to the road users, has to be viewed from a real-time perspective. These real-time applications pose also new requirements and challenges to data management solutions supporting the traffic information life cycle. The notion of *data streams* has been introduced especially to address the needs of applications processing high-speed, real-time data (called *data stream applications* in the remainder). In contrast to other data-intensive large-scale applications, e.g., applications using data warehouses, data stream applications deal with unbounded and continuous flows of data (data streams) such as readings from traffic sensors. To keep up with the flow, operations can only process a recent subset of the data. Furthermore, in data stream applications, time is an important factor as recent data is rated more valuable than older data. It seems obvious that for many tasks in the transportation domain data stream applications are well suited. *Data Stream Management Systems* (DSMS) have been proposed to support the specific properties of data streams and data stream applications from various domains. Depending on the system, they realize many or all of the eight requirements of a real-time stream processing system postulated in [Stonebraker et al. 2005], such as an active behavior (event-driven processing) or the ability to deal with unordered and missing data. Especially, all of them provide the possibility of defining *continuous queries*, i.e., queries which are constantly evaluated over the streams of interest.

To be able to analyze data in real-time, the data management system for traffic applications should contain a DSMS for the basic operations on data streams, but, when also mining should be employed, it should also provide a *data stream mining framework* [Domingos and Hulten 2003] for efficient online analysis of raw traffic data streams. Data mining methods are well-known techniques for deriving new information from base data, but the analysis is usually done 'offline' on stored data. These methods have to be adapted to data streams such that they are able to process the data within a limited time window and 'online' without requiring data to be stored.

Data stream management systems and data stream mining algorithms have been studied for some years [Babcock et al. 2002, Domingos and Hulten 2003], but there are only few approaches which apply these techniques to traffic management applications (e.g., [Liu et al. 2006]). Linear Road [Arasu et al. 2004] is a well known benchmark in the area of DSMS, but it focuses on the performance of the DSMS and not on the characteristics of the traffic application. A contest at the International Conference on Data Mining (ICDM) in 2010 [Wojnarski et al. 2010] evaluated the quality of data mining algorithms for traffic applications. However, the challenge focused only on one particular traffic situation and did not take into account different parameters of the

DSMS or the traffic application. Traffic situations can also be described by the concept of events, defined as "occurences within a particular system or domain" [Etzion and Niblett 2011]. Event processing is strongly related to stream management. In fact, event processing applications can be implemented using a stream management system where each stream consists of a set of events. However, data stream management systems are more general and can be also used to process other kinds of streams [Etzion and Niblett 2011]. In event processing two programming styles can be distinguished [Etzion and Niblett 2011]: stream-oriented and rule-based programming. The stream-oriented programming style corresponds to the stream management concepts we utilize in this work.

To develop effective traffic applications, a thorough evaluation of the techniques employed is necessary. The output quality of the traffic application is not only dependent on the choice of the DSMS and of the mining algorithm; there are many parameters (e.g., the window size in data stream queries, the traffic volume, features of the road network) that have to be considered to achieve a certain quality level in the derived traffic information.

Hence, our contribution in this paper is an *evaluation framework for traffic applications* based on data streams. The framework is composed of a traffic simulation, a data stream management system and a data stream mining component. Using the evaluation framework, we are able to implement specific traffic applications, such as hazard detection or traffic state estimation. At the same time, we can test the efficiency and effectiveness of a particular traffic application and investigate the influence of multiple parameters of the DSMS, the data stream mining algorithms, and the traffic environment. Furthermore, in this article we present the results of applying the framework to evaluate stream mining algorithms and parameter settings for two traffic applications: queue-end detection and traffic state estimation.

We developed the framework in the course of the project Cooperative Cars[1] (CoCar) and its follow-up Cooperative Cars eXtended (CoCarX). The aim of the project is to investigate and implement a hybrid Car-to-X communication infrastructure and corresponding applications based on cellular networks and pWLAN. The vehicles in the scenario send, amongst others, hazard warning messages. Our data management system receives these messages in a data stream and uses them to derive other traffic information, such as the position of a queue-end.

An earlier version of the framework and preliminary results have been presented in [Geisler et al. 2010b]. We have continued our research on the evaluation framework and present more results for the aforementioned traffic applications, but also more details about the system itself in this paper. Especially, we did extensive tests with several data stream mining algorithms and multiple runs with varying parameters for the traffic situation and the data management system.

The paper is structured as follows. The Section 2 gives a brief overview about data streams and data stream mining. Section 3 presents the architecture of our evaluation framework. To show the feasibility and effectiveness of our evaluation framework, we implemented two case studies: the first is a queue-end detection scenario and presented in Section 4, the second study addresses the problem of traffic state estimation and its results are detailed in Section 5. Related work is discussed in Section 6 before we conclude the paper in Section 7 where we also point out future work.

[1] http://www.aktiv-online.org/english/aktiv-cocar.html

2. Background

In this section we sketch important foundations used in the remainder of this paper. We start off with definitions of data streams and related concepts, such as windows. After we briefly discuss data stream management systems and query languages, we review data stream mining techniques.

2.1. Data Streams

A data stream is commonly viewed as a continuous and unbounded flow of data elements coming from a source, without any guarantee of completeness or order [Babcock et al. 2002]. There exist different data models for data streams defined in the context of specific DSMSs. In this paper we adhere to the relational model and the well known multiset or bag semantics for data streams introduced in [Arasu et al. 2006]. We reformulate the definition of a data stream as follows:

Definition 1. *(Data Stream)*
A data stream S is a (possibly infinite) multiset of data stream elements (s, τ), where $\tau \in T$ is a timestamp attribute with values from a monotonic, infinite time domain T with discrete time units $\in \mathbb{N}$. s is a set of attributes $(A_1, A_2, \ldots, A_n)$ with domains $dom(A_i), 1 \leq i \leq n$, constituting the schema of S. A stream starts at a time τ_0. $S(\tau_i)$ denotes the content of the stream S at time τ_i.

In line with other works, we do not consider the ordering timestamp attribute (the attribute which determines if a data stream element is more recent than another one) as part of the schema of the stream. Also note, that the time domain is not strictly monotonic, i.e., we allow stream elements with the same timestamp. We also follow a *time-driven* model, i.e., queries are evaluated at a certain point in time [Jain et al. 2008].

Example 1. In our case studies we receive the messages sent by the equipped vehicles as a data stream. The simplified schema of the stream `CoCarMessage` looks as follows,

$$CoCarMessage(Timestamp, AppID, Speed, Acceleration, Latitude, Longitude)$$

where `AppID` represents the message type, such as an emergency brake message, `Longitude` and `Latitude` represent the position of the vehicle and `Speed` and `Acceleration` represent the current values for these attributes of the vehicle. In the example, the timestamp is *explicit*, i.e., it has been defined by the application creating the tuple, in this case the in-vehicle software. When a DSMS is used to process the data, *implicit* timestamps can be assigned to the tuples when they enter the DSMS, i.e., an additional timestamp attribute is added to the stream element.

Example 2. In the `CoCarMessage` data stream, the tuples have an explicit timestamp field as shown in Example 1. When they enter a DSMS using implicit timestamps (this is the case in our case studies) the stream elements look as follows, where `Timed` is the implicit and `Timestamp` is the explicit timestamp:

$$CoCarMessage(Timed, Timestamp, AppID, Speed, Acceleration, Latitude, Longitude)$$

It clearly depends on the target stream application which type of timestamp is useful. A disadvantage of explicit timestamps is, though, that they do not ensure a correct non-decreasing ordering of tuples due to, e.g., problems with the transmission channel or issues with the sources. With implicit timestamps the processing system dictates the order of tuples. Therefore, in DSMS often implicit timestamps are used. Some DSMS also allow both types, implementing ordering mechanisms such as heartbeats in the STREAM system [Srivastava and Widom 2004].

A distinctive property of data streams is their potential unboundedness. Especially when it comes to querying streams, common operators from relational queries have a problem. *Stateless* operators, such as projection or selection operators can produce a result immediately with each new stream element. In contrast, *stateful* or *blocking* query operators, such as joins or aggregates, rely on finite sets to operate on. For example, the calculation of a maximum will block until the end of the input data has been reached. The most common solution to the unboundedness problem is the use of windows. A *window* limits the stream at the current point in time to a defined finite set of elements. The semantics of a query on a data stream is defined by the semantics of a corresponding relational query, in which each stream is replaced with a relation that holds the content of the stream at a specific time [Golab and Özsu 2010]. Some DSMS actually employ this model for query evaluation, e.g., in Global Sensor Networks (GSN, [Aberer et al. 2006]) streams are translated to temporary relations before a query is evaluated. GSN is also used in the implementation of our framework.

The type of window can be described according to [Patroumpas and Sellis 2006] by its measurement unit, the edge shift, and the progression step. The measurement unit can be either a number of x time units (*time-based window*) or tuples (*tuple-based window*) declaring that the elements with timestamps within the last x time units or the last x elements are valid for the window at the point in time of the query. The edge shift describes the motion of the upper and lower bounds of the window. Each of them can either be fixed or moving with the stream. For example, in the most common variant, the *sliding window*, both bounds move, while for a *landmark window* one bound is fixed and one is moving. Finally, the progression step defines the intervals between two movements of a window. This again can either be time- or tuple-based, e.g., the window can move every 10 seconds or after every 100 arrived tuples. Therefore, we define a time-based window of size l_T as follows (adapted from the definition in [Patroumpas and Sellis 2006]):

Definition 2. *A* time-based window W_{l_T} *(with window size* $l_T \in \mathbb{N}$*) over a stream* S *at time* $\tau_i \in T$ *is a finite multiset of stream elements with*

$$W_{l_T}(S(\tau_i)) = \{(s_k, \tau_k) \,|\, (s_k, \tau_k) \in S, \tau_i - l_T \leq \tau_k \leq \tau_i, \tau_k \geq \tau_0\}.$$

The definition for a tuple-based window of size l_N is similar:

Definition 3. *Let* $S(\tau_i) = \{(s_0, \tau_0), \ldots, (s_n, \tau_n)\}$*,* $\tau_i \geq \tau_n \geq \tau_0 \wedge \tau_j \geq \tau_{j-1} \forall j \in \{1, \ldots, n\}$*, be the content of stream* S *at time* τ_i*. Then a* tuple-based window W_{l_N} *(with window size* $l_N \in \mathbb{N}$*) over stream* S *at time* $\tau_i \in T$ *is a finite multiset of stream elements with*

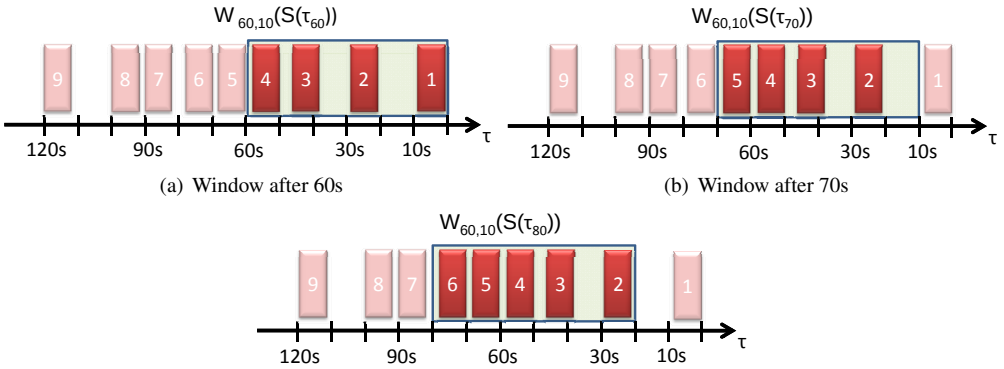
$$W_{l_N}(S(\tau_i)) = \{(s_k, \tau_k) \,|\, (s_k, \tau_k) \in S, n - l_N \leq k \leq n, \tau_i \geq \tau_k \geq \tau_0\}.$$

Furthermore, we define a sliding time-based window as:

Definition 4. *A* sliding time-based window *with window size* $l_T \in \mathbb{N}$ *and slide value* v *over a stream* S *at time* $\tau_i \in T$ *is a finite multiset of stream elements with*

$$W_{l_T,v}(S(\tau_i)) = \{(s_k, \tau_k) \mid (s_k, \tau_k) \in S, \exists j \in \mathbb{N} : \tau_0 + j \cdot v \leq \tau_i, \tau_0 + (j+1) \cdot v > \tau_i, \tau_i \geq l_T \\ \tau_0 + j \cdot v - l_T \leq \tau_k \leq \tau_0 + j \cdot v, \tau_k \geq \tau_0\}.$$

Example 3. In our traffic example we want to retrieve the number of CoCar messages with `speed` greater than 30 km/h from the last minute every 10 seconds. We realize this by defining a sliding time-based window $W_{l_T,v}$ of size $l_T = 60$ s and with a sliding step of $v = 10$ s, over the `CoCarMessage` stream. In Figure 1(a) the content of the window after one minute is shown. The window contains the elements 1, 2, 3, and 4. After 10 more seconds the window slides (i.e., the value of j in Definition 4 is increased from 6 to 7), element 1 is dropped from the window and element 5 is added (Figure 1(b)). After another slide after 10 seconds element 6 is added to the window (Figure 1(c)).



(a) Window after 60s

(b) Window after 70s

(c) Window after 80s

Figure 1: Example of a sliding window with size of 60 s and sliding step of 10 s

2.2. *DSMS and Data Stream Query Languages*

Data Stream Management Systems (DSMS) have specific properties to process data streams in an efficient way. Especially, their active behavior, i.e., the reaction to incoming data and executing continuous queries on it, is very suitable for event detection. DSMS have to address all problems which are inherent to data streams, such as unordered, delayed, bursty, or missing data. There exist several DSMS research prototypes, such as STREAM [Arasu et al. 2003], Borealis [Abadi et al. 2005], StreamMill [Bai et al. 2006], or GSN [Aberer et al. 2006]. Also various commercial products by global players have been released, e.g., IBM InfoSphere Streams[2], Microsoft StreamInsight[3], or StreamBase[4]. We will not go into further details of DSMS architectures, but the interested reader is referred to an overview of DSMS architectures given in [Ahmad and Çetintemel 2009]. The query processing as well as the query languages in DSMS have to

[2] `http://www.ibm.com/software/data/infosphere/streams/`
[3] `http://www.microsoft.com/sqlserver/2008/en/us/r2-complex-event.aspx`
[4] `http://www.streambase.com`

fulfill specific requirements to account for stream properties. First of all, the semantics detailed in Section 2.1 has to be implemented to represent streams and windows on streams in the query language. Furthermore, in many applications there is the need of combining data from streams with data from static relations [Cherniack and Zdonik 2009]. Dealing with both, streams and relations, in queries, there is clearly a need to let the language be closed under streams [Cherniack and Zdonik 2009], to enable nesting and optimization of queries. Other required functionality may be means to analyze data with data mining algorithms [Thakkar et al. 2008]. We will briefly explain the specifics of data stream mining algorithms in Section 2.3. Many DSMS provide query languages which are a variant of the SQL standard and introduce new query operators, such as windows, to enable continuous queries on streams (see, for example, [Arasu et al. 2006]).

2.3. Data Stream Mining

Data stream mining algorithms provide the ability to analyze the data while it is streaming by. Achieving this and at the same time being efficient, the algorithms have to fulfill the properties determined in [Domingos and Hulten 2003]. These properties include the one-pass property (every record can only be seen once), the amount of memory for the model has to be limited, the amount of time to process each element has to be constant and the algorithm should be able to adapt to concept drift, i.e., it should update the model whenever the process creating the data or the underlying data distribution changes [Tsymbal 2004]. Variants of mining algorithms for data streams from different parts of the field have been introduced, such as clustering algorithms for uncertain data streams in [Aggarwal and Yu 2008] or finding frequent itemsets, surveyed in [Cheng et al. 2008].

In this article, we will concentrate on classification algorithms as we cast the addressed problems of queue-end detection and traffic state estimation as classification tasks. Classification is the problem of assigning one of a set of class labels to a data record. A classifier can, for example, be built with a (supervised) learning technique using a set of (labeled) training data. The output of the learning phase, the classifier, is represented by a model which is utilized to predict a class for unknown data records [Han and Kamber 2006]. Commonly, a set of training data, used to build the model, is carefully segregated from a set of test data which is used to evaluate the accuracy of the model (percentage of correctly predicted elements). Unlike in the traditional setting, in data stream mining the amount of training and test data is not a problem – data superabounds. However, algorithms cannot rely on having the complete training set in memory. Therefore, new training and evaluation methods to work on streams have to be found. Two procedures for stream mining have proven suitable [Bifet et al. 2010a;b]. The *Interleaved Test-Then-Train* or *Prequential* approach uses each incoming stream element first for testing the model and then for training, allowing a very fine-granular evaluation. In the prequential method accuracy can also be evaluated only over classification results of a window containing the last n elements [Gama et al. 2009]. The *Holdout* method periodically evaluates the model with a set of elements held back, giving a snapshot of the current accuracy of the model. We used the first method in our framework. In the following we will briefly explain some stream classification algorithms.

Decision trees can be considered as one of the most popular and also a very efficient type of classification algorithms. Domingos and Hulten [2000] proposed a well-known decision tree learner implementation, termed Very Fast Decision Trees (VFDTs). VFDTs are based on Hoeffding Trees, which use a statistical measure called Hoeffding bound. The bound determines, how many examples are required at each node to choose a split attribute. It guarantees, that with a given probability the same split attribute is selected as the attribute which would have been

chosen if all training elements were known already. The algorithm first determines the two best attributes by using a relevance measure such as information gain or the Gini index. Afterwards, if the difference between the two attributes' relevance values excels the calculated Hoeffding bound for the examples seen so far, the best attribute is chosen for the split.

This algorithm has been further extended to Concept-adapting VFDTs in [Hulten et al. 2001]. The algorithm detects concept drift in time-changing data streams and grows alternative branches for subtrees with an insufficient accuracy. When the alternative subtrees deliver better accuracy values than the old ones, they substitute them in the tree. There have also been approaches which use ensemble techniques for learning. For example, [Hashemi et al. 2009] propose a one-versus-all-classifier technique where for each class a binary classifier is trained, i.e., the tree only decides between the class it is specialized on and all other classes. For each element all classifiers decide if the element is in their own class and the classifier with the highest confidence value gets the vote. Other well-known ensemble techniques are the online bagging and boosting algorithms presented in [Oza and Russell 2001, Oza 2006].

Besides decision trees other classification schemes have been adapted to or have been newly implemented for data streams. These include Support Vector Machines, e.g., presented in [Laskov et al. 2006] or fuzzy-rule classification, e.g., in [Angelov and Zhou 2008]. Furthermore, Gama and Rodrigues pointed out that neural networks are well suitable for data streams without any changes, using stochastic sequential training [Gama and Rodrigues 2009].

3. Architecture and Data Stream Processing

In this section, we first describe the architecture of our evaluation framework, and then we go into detail on data stream components in the DSMS which are used to process and mine the data.

3.1. Data Stream-based Fusion Architecture

According to [Hoyer 2003] data fusion in transportation can be defined as "a process in which signals or data of sensors are automatically detected, associated, combined, and estimated." The evaluation framework proposed in this paper is based on a data management and fusion architecture which we designed and implemented in the CoCar project along the data fusion process [Geisler et al. 2009]. The main constituents of the architecture are first of all the mobile sources which deliver the traffic data to be fused. Second, the data is received by a data stream management system which processes and integrates the data. Third, to analyze the data and derive new traffic information, we integrated a data stream mining framework into the DSMS. Last but not least, we use a spatial database to speed up working with geospatial data. We detail these components in the following. An overview of the architecture with its main constituents is depicted in Figure 2.

3.1.1. Data Sources

The architecture is designed to enable the use of multiple data sources which are integrated in the data stream management system (DSMS). The data sources are one parameter which is very interesting to evaluate in a traffic application. General questions include which data sources are sufficient for which traffic application and what is the effect on the resulting data quality, when additional data sources are integrated. In this work, special emphasis is put on data sources of mobile detection as they offer several advantages over stationary detection. One goal of our work

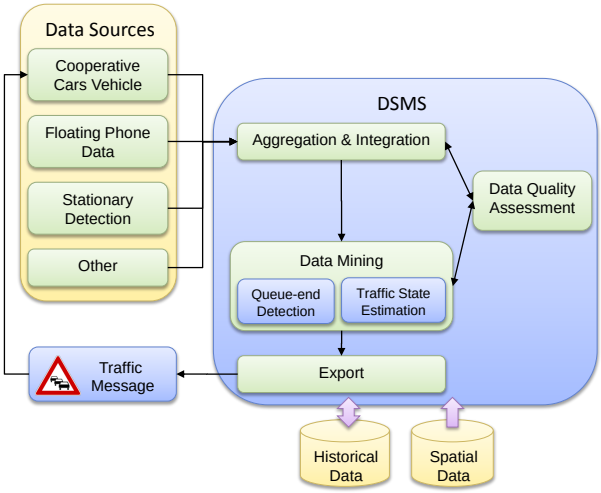


Figure 2: The overall architecture

is to find out to which extent and within which limits mobile data sources can be used for traffic applications, such as derivation of traffic data [Geisler et al. 2010a], hazard detection, or traffic state estimation. For reasons of reproducibility, controllability and creation of sufficient amounts of data, we use the traffic simulation software VISSIM by PTV AG[5] to emulate the creation of data from mobile and stationary sensors.

In the CoCar project, one important data source is, of course, the CoCars sending event-based messages in case of hazards to warn other vehicles. We create CoCar messages by using the VISSIM simulation for the following events: a vehicle braking very hard and a vehicle turning on its warning flashers. For traffic state estimation we introduced another type of message, which is send periodically (not event-based) by the vehicles and which only contains basic information, such as position and speed. Furthermore, since we apply supervised mining algorithms, the traffic simulation also needs to create ground truth data for the different traffic applications. For example, for the queue-end detection, the exact position of queue-ends in the simulation is determined and also sent to the DSMS. In the case studies all messages are transmitted via TCP. In a real setting, mobile communication protocols such as UMTS (Universal Mobile Telecommunication Systems) or LTE (Long-Term Evolution) will be used. The data included in the messages will be detailed in Section 4.

The second important data source is Floating Phone Data (FPD). FPD are anonymously collected positions of mobile phones. From these positions other traffic data, such as the vehicle speed, can be derived. Other data sources can be data from stationary detection sensors. The data can be pushed to or pulled from the DSMS from the corresponding data source. In our case studies, we focus on the CoCar messages only as we want to analyze the usefulness of CoCar messages and the influence of parameters, such as required equipment rates, for certain traffic applications.

[5] `http://www.ptv.de`

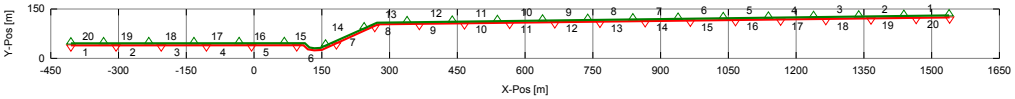


Figure 3: Two links subdivided into sections

3.1.2. Data Stream Management System

We use the Global Sensor Network[6] (GSN) system [Aberer et al. 2006] for data stream management. GSN provides a flexible, adaptable, distributable, and easy to use infrastructure. It wraps functionality required for data stream processing and querying around existing relational database management systems, such as MySQL or Microsoft SQL Server. We decided to use the MySQL in-memory database to ensure efficient processing. The main concepts in GSN are wrappers and virtual sensors. In GSN, we provide a wrapper for each data source to receive the data. Each further processing step is encapsulated into a virtual sensor which creates the output data by a query over the input data. For data stream mining, we integrated the data stream mining framework Massive Online Analysis (MOA)[7] into the DSMS. We implemented a virtual sensor for GSN which applies a data stream mining algorithm on aggregated sensor data and outputs the results as a data stream to the DSMS. The virtual sensor is generic, i.e., the type of the classifier algorithm and the corresponding configuration can be defined in a configuration file.

The DSMS exports the derived information and sends it as messages over the CoCar infrastructure to the connected road users, e.g., by Geocast (i.e., only into a certain geographic area). The result can, for instance, be a message announcing a queue-end at a certain position. The reception of messages by the vehicles might influence their driving behavior and the influence of this behavior on the corresponding traffic application can be analyzed. However, this requires a deep understanding of how drivers react to certain warning messages based on social, cognitive, and empirical studies which are out of the scope of our research. In our evaluation framework, we also can export statistics for evaluation, e.g., number of classified instances so far, utilizing the export functionality of GSN.

3.1.3. Spatial Database

All traffic applications have spatial characteristics per definition. *Spatial databases* have been introduced to ease and speed up the work with spatial data using special data types and functions. In our architecture, we store and query the road network at hand by using the spatial functionality of Microsoft SQL Server 2008[8]. We export the roads (also called *links* in the following) from the traffic simulation and store them as curve objects in the database. In the database, we divide the links into sections of equal length, e.g., sections of 100 m length each. Sections are the units of the road network we will work with in the scenarios, e.g., determine for each section if a queue-end is located on it or determine the traffic state for each section. Figure 3 shows an example for two links (more precisely, a link is one direction of a road) subdivided into sections. In both of our case studies we investigated if the section length has an impact on the mining accuracy and we present the results in Section 4.1 and Section 5.1.

[6] `http://sourceforge.net/projects/gsn/`

[7] `http://www.cs.waikato.ac.nz/~abifet/MOA`

[8] Other DB products provide similar functions for spatial data, thus, we are not limited to this particular product.

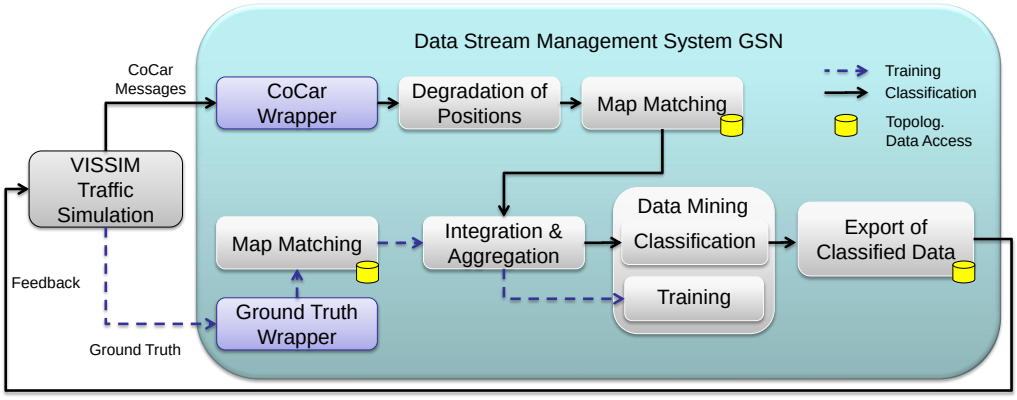


Figure 4: Overview of the data stream processing components in the DSMS

Each time we have to localize a vehicle on the network (a process termed *Map Matching*), we issue a query to the spatial database to find the section the vehicle is driving on. The Map Matching and usage of spatial data in our framework is explained in detail in Section 3.2.

3.2. Data Stream Processing

We now have a closer look at how the data is processed inside the DSMS. An overview of the data flow in the system and the implemented components is depicted in Figure 4.

3.2.1. Data Stream Reception

As we have detailed in the previous section, GSN provides two structures to work with data streams: wrappers and virtual sensors. The wrappers manage the connections to the data sources, receive the data and convert it to GSN stream elements, which are in fact relational tuples. In our case studies, the CoCar messages and ground truth messages are produced by the traffic simulation and sent via TCP to the DSMS. We implemented a wrapper for each type of message.

Using Definition 1 for data streams, the schema of the stream produced by the wrapper receiving the CoCar messages looks as follows (we only include the attributes which are of importance for the case studies):

$$CoCarMessage(Timed, TS, Lat, Lng, Speed, Accel, ApplicationID)$$

where `Timed` is the implicit timestamp given by GSN, `TS` is the explicit timestamp, `Lat` is the latitude of the vehicles position, `Lng` is the longitude of the position, `Speed` is the speed of the vehicle at that time, `Accel` is the acceleration and `ApplicationID` denotes the type of the message, which can either be an Emergency Brake Light message (EBL), a Warning Light Announcement message (WLA) or a Vehicle Probe Data message (VPD, a message which is sent out periodically by the CoCar). The ground truth messages differ slightly in their schema depending on the case study. In the queue-end detection scenario they contain the position of a queue-end and their schema is:

$$QueueEnd(Timed, TS, Lat, Lng)$$

where `Timed` and `TS` are defined as before, and `Lat` and `Lng` represent the position of the queue-end. While the ground truth messages for the queue-end still have to be matched to a link and

section of the network, in the traffic state estimation scenario the traffic simulation already added the IDs of link and section to the ground truth messages:

StateEstimation(*Timed*, *TS*, *LinkID*, *SectionID*, *TrafficState*)

where `Timed` and `TS` are defined as before, `LinkID` and `SectionID` denote the section and its corresponding link for which the true traffic state is reported and `TrafficState` is a class value representing true traffic state in this section.

3.2.2. Degradation and Map Matching

One common issue in mobile data detection is, that the measured positions contain some error whose amount depends on the used positioning technique. This error influences the accuracy of traffic information which has been shown, e.g., in [Geisler et al. 2010a], and therefore, has to be taken into consideration. In case of the CoCars messages we can assume GPS positioning accuracy. To approximate reality as close as possible the exact positions in the messages created by VISSIM have to be degraded. To that end, the stream elements created from the CoCar messages are forwarded to a degradation virtual sensor. We use a normal distribution to model the error. When working with real data sources this step is obsolete, of course. The accurate positions are replaced by the degraded positions in the stream elements and forwarded – the schema and all other data remain unaltered.

The next virtual sensor matches the positions of the CoCar messages to the road network, which is termed *Map Matching*. Map Matching is the process of finding the closest link and point to the position where the road user actually is located. We use a simple Map Matching technique, where the link and the section with the shortest perpendicular distance to the measured point are selected. More sophisticated methods also consider the trajectory of a vehicle and the topology of the network. For a detailed survey on Map Matching see for example [Jensen and Tradisauskas 2009, White et al. 2000]. The CoCar messages do not contain any identifying information about the vehicle sending the message because of privacy reasons. Without information about the trajectory of the vehicle, we can only utilize this simple Map Matching technique.

The Map Matching also contributes to a realistic scenario as it introduces an error common in traffic applications, too. For the positions in the ground truth messages of the scenarios, it is mandatory that they are accurate and therefore, these are not degraded. However, to determine the links and sections the ground truth positions are located on, in the queue-end scenario the corresponding stream elements are forwarded from the wrapper to a second map matching sensor. This sensor works in the same way as the CoCar Map Matching sensor. It introduces no error, because the positions lie exactly on the sections and are matched precisely. In the Map Matching sensors, we add this new information of link and section number to the stream element. Hence, the schemas of the forwarded stream elements have to change:

CoCarMessage(*Timed*, *TS*, *Lat*, *Lng*, *Speed*, *Accel*, *ApplicationID*, *LinkID*, *SectionID*)

The ground truth messages for the queue-end detection scenario also have to be map matched in another virtual sensor. The attribute `HasQueueEnd` will be always 1 as the stream will only contain tuples for the sections with a queue end.

QueueEnd(*Timed*, *TS*, *Lat*, *Lng*, *LinkID*, *SectionID*, *HasQueueEnd*)

3.2.3. Aggregation and Integration

After the position of each message has been matched to a section, we need to prepare the data for data stream mining. The overall goal of the mining process is to determine a class for a section, characterized by a set of data. In our approach, we aggregate CoCar messages with positions lying on a particular section for a certain time window. This means, we calculate the following parameters from the CoCar data stream elements for one section and a time window over the last x time units:

- average speed (`AvgSpeed`)
- average acceleration (`AvgAccel`)
- number of Emergency Brake Light messages (`EBLNo`)
- number of Warning Light Announcement messages (`WLANo`)

When including the Vehicle Probe Data (VPD) messages, the number of these messages is also determined for the section at hand. Virtual sensors are configured each by an XML file. In GSN, there are two levels of continuous queries defined in these XML files: firstly, the required data streams currently active in the GSN system can be queried; secondly, an overall query for joining the results of the data stream queries can be defined. This means, the aggregation and integration can be done in one virtual sensor. To complete the training dataset for the data stream mining, we have to join the aggregated data with the ground truth stream elements, which define the true class value of each dataset. Both streams are joined over the section and the time window. The join is obsolete if no training is performed.

We now formulate the queries used to aggregate and integrate the data from the CoCar messages stream and the ground truth streams using the semantics from Section 2 and the extended relational algebra semantics [Garcia-Molina et al. 2009]. We give here the example for the queue-end detection. For the traffic state estimation the queries look similar. First, we define the queries retrieving the data from the windowed two streams:

$$\begin{aligned} vQueueEnd &= \pi_{LinkID,SectionID,HasQueueEnd}(\\ &\quad \gamma_{LinkID,SectionID,HasQueueEnd}(W_{120,10}(QueueEnd))) \\ vCoCar &= \pi_{Speed,Accel,ApplicationID,SectionID,LinkID,TS}(W_{120,10}(CoCarMessage)) \end{aligned}$$

The γ-operator in this extended relational algebra expression performs the grouping over the input relation. In this example, the sliding window over the `QueueEnd` stream is grouped by `LinkID`, `SectionID`, and `HasQueueEnd`. In GSN, the results of these queries are represented by views and therefore, can be used as relations in the integrating query. The integrating query for the queue-end example looks as follows in relational algebra (note, that we do not need to specify a window here anymore):

$$\begin{aligned} &\pi_{AvgSpeed,AvgAccel,HasQueueEnd,WLANo,EBLNo,LinkID,SectionID}(vQueueEnd \\ &\quad \bowtie(\gamma_{SectionID,LinkID,AVG(Speed)\rightarrow AvgSpeed,AVG(Accel)\rightarrow AvgAccel}(vCoCar) \\ &\quad \bowtie(\sigma_{ApplicationID='WLA'}(\gamma_{SectionID,LinkID,Count(*)\rightarrow WLANo}(vCoCar)) \\ &\quad \bowtie(\sigma_{ApplicationID='EBL'}(\gamma_{SectionID,LinkID,Count(*)\rightarrow EBLNo}(vCoCar)))))) \end{aligned}$$

This query results in the creation of stream elements which contain the desired aggregated values per section and the ground truth class for training. The stream elements are now ready to

be fed into the data stream mining algorithm. The resulting schema looks as follows:

$$MiningElement(Timed, AvgSpeed, AvgAccel, HasQueueEnd, WLANo, EBLNo, LinkID, SectionID)$$

3.2.4. Data Stream Mining

We already mentioned, that we only use classification in our framework. As depicted in Figure 4, we can setup the scenario according to the two classical supervised learning modes: training and classification. If a classification without training is required, the obsolete virtual sensors can easily be removed. As already mentioned, we integrated the data stream mining framework MOA into GSN by encapsulating it into a virtual sensor. MOA is based on the well-known mining framework Weka[9]. In the mining virtual sensor, the incoming data stream elements are converted into MOA compatible instances.

After the mining algorithm classified an instance, the statistics of the classifier are updated and the classifier is trained on the instance. The statistics of the classifier comprise values of a *confusion matrix*. For N classes a confusion matrix contains NxN values for each combination of predicted class and true class. Then, a corresponding CoCar message is generated and can be exported, e.g., archived in a database or sent to the traffic simulation, where it can be used to visualize the corresponding event. The stream elements produced by the mining sensor have the following schema for queue-end detection:

$$ClassifiedElement(Timed, AvgSpeed, AvgAccel, HasQueueEnd, WLANo, EBLNo, LinkID, SectionID, ClassQueueEnd)$$

where `ClassQueueEnd` is the predicted class.

4. Queue-End Detection

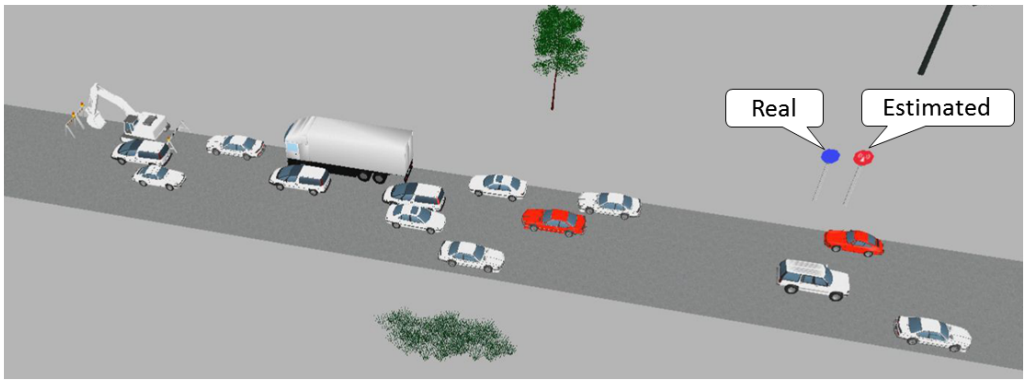
To show the feasibility and effectiveness of our evaluation framework, we set up two case study scenarios. The scenario described in this section aims at the detection of queue-ends. In our traffic simulation network, a road consists of two links, one for each direction. A link is divided into sections as detailed in Section 3. For each section it will be determined, if it contains a queue-end or not. The scenario aims at investigating the influence of multiple parameters on the detection accuracy. As a source for the traffic data we use CoCar messages created by VISSIM as described in Section 3.1. We detail the scenario and evaluation setup in this section and present results from the evaluation.

As mentioned earlier, we use a traffic simulation to create the traffic data. In this case study, the simulation operates on a simple road network consisting of a highway of 5 km length, i.e., two links with two lanes each. The speed limit is unrestricted. The links are almost straight, with one sharp turn as depicted in Figure 5(a). One link contains a hazard (a construction site with an excavator) narrowing the street to one lane, shown in Figure 5(b). A reduced speed area encloses the construction site, restricting the maximum speed to 70 km/h.

Since we employ learning, besides the network and the CoCar messages, we also need to determine the *correct* class for a section (does it contain a queue-end?) to train the mining algorithm, i.e., we need the ground truth. Hence, we added queue counters in the VISSIM road

[9] `http://www.cs.waikato.ac.nz/~ml`

(a) Simulation track



(b) Hazard on the track

Figure 5: The scenario track and a hazard

network, which are able to measure the length of a queue starting from their own position. A vehicle starts queuing when it is slower than 15 km/h and has a maximum distance of 20 m to the vehicle in front. It stops queueing if it is faster than 30 km/h. The queue lengths are measured every ten seconds and if a queue-end is detected, a corresponding message is created which contains the position of the queue-end. During the simulation, the CoCar and queue-end messages are sent via TCP to the server hosting GSN. The mining sensor in this scenario uses currently the Hoeffding Tree classifier included in the MOA framework, which is "state-of-the-art for classifying high speed data streams" [Bifet and Kirkby 2009a]. The classifier has to decide according to the dataset at hand, whether a section contains a queue-end or not. To test the accuracy of the algorithm, we use the Interleaved Test-Then-Train approach. This maximizes the use of the examples available and it allows an accuracy analysis on a very fine-granular level [Bifet and Kirkby 2009a] as the training examples are first used for testing and then for training. The statistics of the tests comprise the number of instances used for training so far and the values for true positives and negatives and false positives and negatives, i.e., we have a 2×2 confusion matrix (we detail the accuracy calculation in the next section). For the configuration of the Hoeffding Tree algorithm, a default set of options has been used, because we want to focus on the evaluation of the influence of varying traffic scenario parameters, such as traffic volumes. In addition, we study also the effect of parameters of the data management system, e.g., the window sizes in the DSMS. In the evaluation of the traffic state estimation scenario we also varied the mining algorithms and their parameters.

After mining, the stream element is enriched with the predicted class and the mining statistics. If the algorithm predicted a queue-end for the section in the classified stream element, the position at the middle of this section is determined in the spatial database and added to the ele-

ment, representing the queue-end. The stream element is converted to a message and sent back to the VISSIM simulation via TCP. In the simulation the estimated queue-end and the correct queue-end are visualized by blue and red traffic signs as shown in Figure 5(b).

4.1. Evaluation and Results

In this evaluation, we focus on the effect of various application-related parameters on the accuracy of the detection method. We analyzed three parameters, which seemed to be most promising in the queue-end scenario: the penetration rate of CoCars, the traffic volume, and the length of the sections. In addition, we studied one system-related parameter: the window size of the queries in the DSMS, i.e., the amount of data from the past which is taken into account by the data stream mining methods. For the queue-end detection time-based windows have been used to select data from the last defined period of time, e.g., the last minute.

For the evaluation, a default configuration of these parameters has been determined. We set the penetration rate to 5%, the traffic volume to 2500 veh/h, and the section length to 100 m. The window size was chosen to be 120 s. To investigate the influence of one parameter, this parameter is varied and all other parameters keep their default value. The following evaluation metrics of the data mining results have been calculated for each run, whereby we define two classes to be identified by the classifier: sections with a queue-end, denoted as *positives*, and sections without a queue-end, denoted as *negatives*. *True positives* are the correctly identified queue-ends, whereas *false positives* are instances erroneously classified as sections containing a queue-end. Correspondingly, *true negatives* and *false negatives* are defined.

- **Overall accuracy:** Determines the ratio of instances with correctly classified classes to the number of all instances classified so far.

$$\frac{\text{True Positives} \ + \ \text{True Negatives}}{\text{Positives} \ + \ \text{Negatives}} \tag{1}$$

- **Sensitivity:** Sensitivity is the ratio of correctly identified queue-ends to the number of all real queue-ends.

$$\frac{\text{True Positives}}{\text{True Positives} \ + \ \text{False Negatives}} = \frac{\text{True Positives}}{\text{Positives}} \tag{2}$$

  In this scenario, we are highly interested in the sensitivity, because it is more dangerous to leave a real queue-end unrevealed as to forecast a non-existent queue-end.

- **Specificity:** Specificity is defined as the ratio of the number of correctly classified sections without queue-end to the number of all Negatives.

$$\frac{\text{True Negatives}}{\text{True Negatives} \ + \ \text{False Positives}} = \frac{\text{True Negatives}}{\text{Negatives}} \tag{3}$$

Each evaluation starts with a new tree, i.e., the classifier has not seen any training instances so far. To ensure comparability, all components in the framework have to work in a reproducible manner, i.e., all randomized elements have to be set to a fixed seed. For example, each traffic simulation run configured with the same parameters is identical, i.e., the raw data produced by the vehicles is always the same as well as the traffic state in each time step. To test the

comparability, two identical evaluation runs with default values have been made before starting the actual tests. The runs had almost identical accuracy gradients and differed no more than 1% in value. Differences in values can be explained by minor time shifts induced by TCP transfer in GSN and differences in time between start of the simulation and start of the GSN system. That means, the time windows might not be at the exact same position in the time line as they had been in the run before and therefore, might not result in identical averages. For the experiments we turned off the degrading to analyze the algorithm in the ideal situation. Comparisons between runs with and without degrading showed that it has an influence, especially on the sensitivity but the overall trend was the same. Each simulation run had a duration of 45 minutes. The average time for mining one element was about 0.6 ms, while the algorithm mined up to 11014 elements per run. In the following, we describe each parameter that has been evaluated before we show and discuss the most interesting results of the experiments.

4.1.1. Window Size

The window size determines how much data of the past is included in the aggregated data of the instances. A good window size would be very close to the average time for which the queue-end stays in one section. For example, if the queue-end is 50 s in section A and 50 s in section B, then with a window size of 100 s (or more) the system would not be able to make a distinction between section A and B. On the other hand, if the window size is too small, the mining algorithm will not get enough information to determine that a queue-end is in a particular section. Based on some initial observations on the time needed for a queue-end to go from one section to the next (we observed times between 10 s and 110 s for the default traffic volume), we decided to vary the window size parameter between 10 s and 300 s. The results for accuracy and sensitivity are shown in Figure 6 and Figure 7, respectively.

The results for accuracy are not helpful to identify a good window size as there are much more examples for negatives than positives, especially for the smaller window sizes (4% positives in the 10 s run, 19% in the 300 s run). Therefore, a window size of 10 s has the best accuracy although there are only very few true positives (cf. Figure 7). The noisy start phase of about 400 s should not be taken into account as the traffic jam has not been established yet and the mining algorithms could not be trained on many positives up to that point. The results for sensitivity show that the bigger the window gets, the better is the sensitivity. This can be explained by the residence time of a true queue-end in the windows. As soon as an actual queue-end is reported for one section, it is included in a window. For bigger windows, an actual queue-end occurs correspondingly longer in the sliding windows (for 300 s it would occur 30 times, for a 10 s window only once). Hence, the number of examples for positives is higher. On the other hand, the specificity is decreasing the bigger the window gets. We decided to keep 120 s as the default window size, at it seems to be a good compromise between sensitivity and specificity and also corresponds to the maximum of the observed duration of a queue-end to stay in one section.

4.1.2. Traffic Volume

The traffic volume is given in vehicles per hour. It is expected that, when the traffic volume increases, more vehicles will be present in one section. This will also lead to a higher volume of CoCars per section and enhance the number of messages per section. Additionally, a queue will grow faster when the amount of traffic is higher, which, in turn, results in a higher rate of vehicles actively braking and switching on their warning flashers. It can be assumed that a higher amount of messages will lead to a more reliable approximation of the actual traffic situation in one section. Therefore, we expect the accuracy to increase when the traffic volume is rising. According

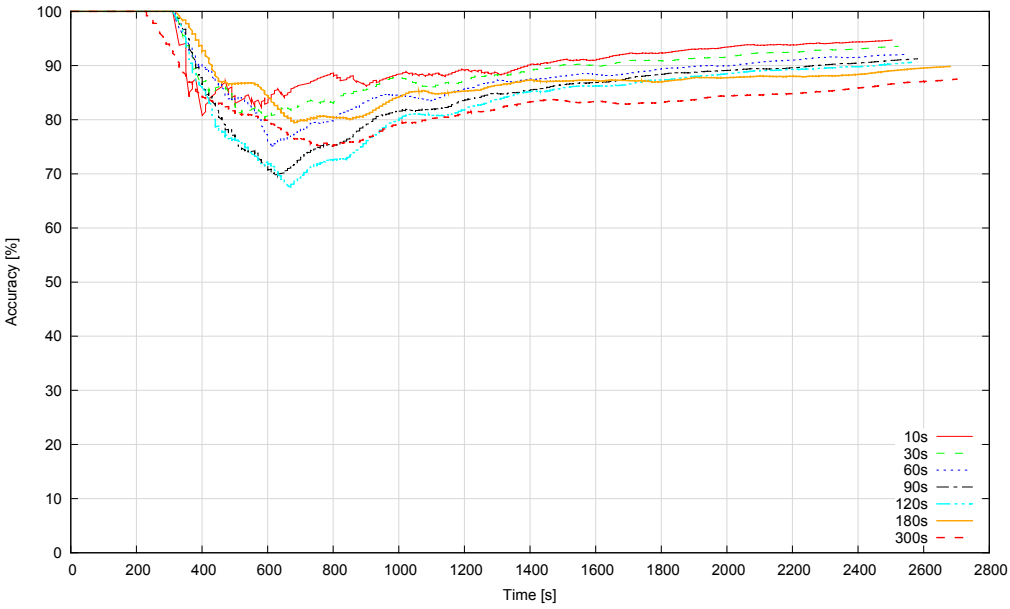


Figure 6: Accuracy for varying window sizes

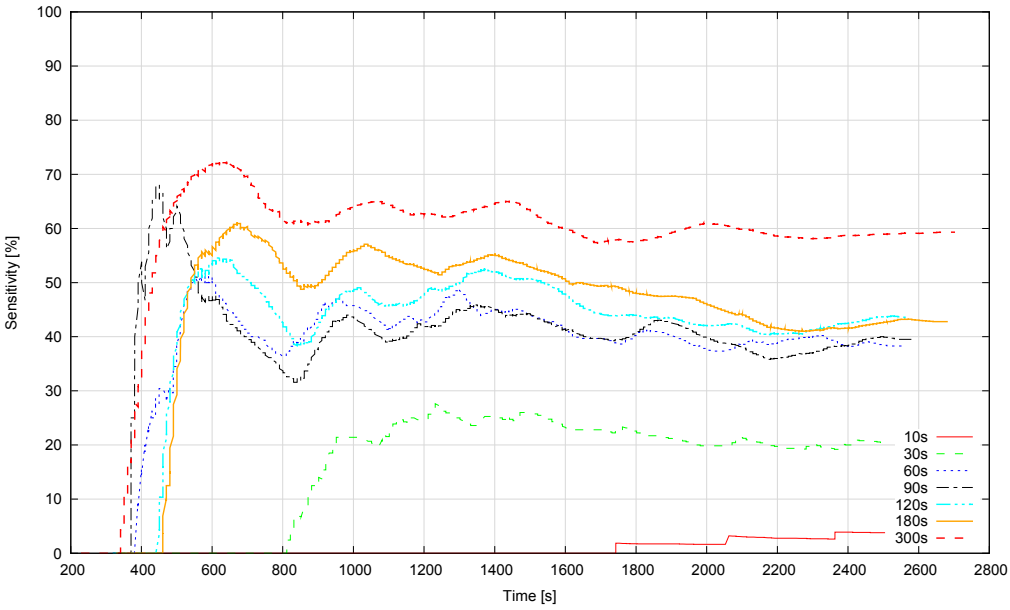


Figure 7: Sensitivity for varying window sizes

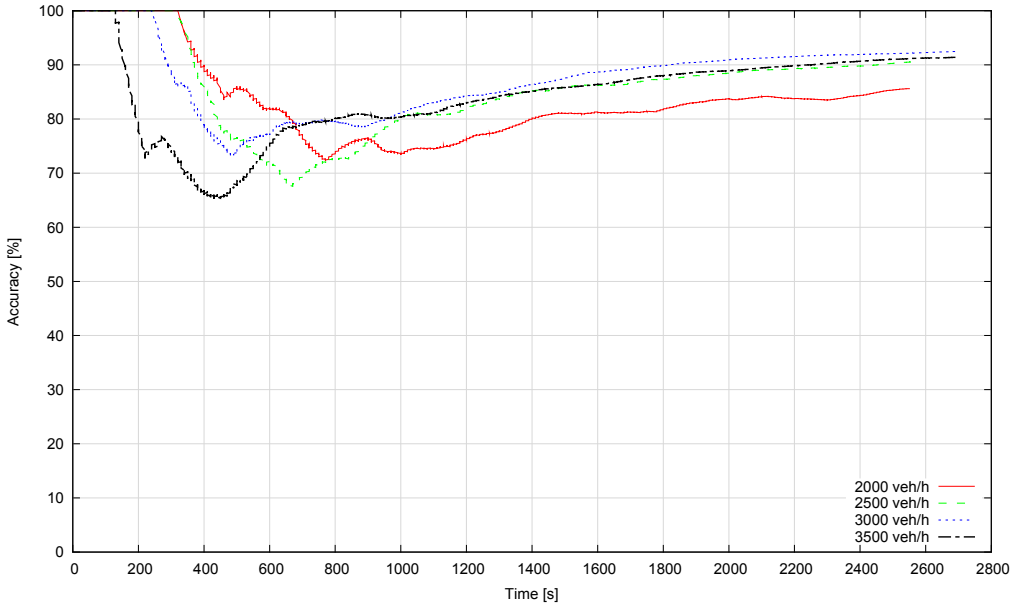


Figure 8: Accuracy for varying traffic volumes

to current statistics published by the German Federal Highway Research Institute on German highways traffic volume averages 1500 veh/h calculated from the daily traffic volume (including the low-traffic night)[10]. However, traffic queues occur at higher volumes during the day. Therefore, we experimented with 2000, 2500, 3000, and 3500 veh/h (additionally, 1500 veh/h did not lead to any queueing in our traffic scenario). The accuracy results depicted in Figure 8 fulfill our expectations: the accuracy rises with an increased traffic volume. Skipping the noisy start phase, it can be observed that the accuracy is above 80% for all tested traffic volumes, even above 90% for a traffic volume of more than 2500 vehicles per hour. As expected, the highest traffic volumes deliver the best results. The number of seen examples increases in each run (which is also dependent on the duration of the run). While in the 2000 veh/h run the classifier trained on 2634 instances, in the 3500 veh/h run the classifier observed 8033 examples which corroborates the assumption, that the number of messages rises with increasing vehicle volume. But the analysis of the specificity and sensitivity evaluation showed, that the biggest portion of the correctly classified instances are constituted by true negatives. The specificity reaches accuracies of 98% at maximum in the 3000 veh/h run and shows the same pattern (rising values with rising volumes), while the sensitivity only reaches about 49% in the 3000 veh/h run and no clear trend in the runs can be observed. It can be seen in the training data that the sensitivity gets rapidly worse when the queue reaches the bounds of the network. The trained algorithm gets in this case contradictory information about how a queue-end looks like in the data (queue-end sections are no longer distinguishable from congested sections, because the queue-end counter reports only the last section of the network as the queue-end) and hence, returns no true positives anymore.

[10] `http://www.bast.de/cln_031/nn_472408/DE/Aufgaben/abteilung-v/referat-v2/verkehrszaehlung/Aktuell/zaehl__aktuell__node.html?__nnn=true&map=0`

This seems to be a sign that the used mining attributes allow a distinction between a section with queue-end and a congested section.

To avoid that the queue reaches the network bounds during simulation time, we extended the links of the network to 10 km length. We experimented with 2500, 3000, 3500 and 4000 veh/h. For these experiments the accuracy and specificity showed no clear trend, but as in the 5 km experiments the 3000 veh/h run was slightly better than the others. For the sensitivity the 4000 veh/h run turned out to be slightly better than the other runs, but also deteriorates at the end of the simulation to the same level than the others.

An analysis of the ratio of the actual positive to negative examples reveals, that with increasing traffic volume and simulation time and hence, increasing queue length, obviously the portion of sections from which messages are received and are containing a queue-end in a time window decreases (from 14% in 2000 veh/h run to 9% in 4000 veh/h run), while the fraction of sections which do not contain a queue-end increases. The latter group are most likely sections with congested traffic, but do not contain the queue-end. This leads to a gradual deterioration of sensitivity from the middle to the end of the simulation time.

4.1.3. Penetration rate

One of the most common and interesting questions for traffic applications based on data from mobile sources is: how many vehicles must be equipped with the technology to deliver acceptable results for an application? Huber revealed in his analysis [Huber 2001], that a general assumption for FCD penetration rates cannot be made, but that it has to be evaluated in the context of the information system, the traffic application, and the required output quality at hand. The penetration rates he identified in the analyzed studies vary between $1 - 5\%$. We took these values as a basis and made experiments with 1%, 3%, 5%, 7% and 10% penetration rate. It is expected that a higher penetration rate leads to a higher accuracy, as it can be assumed that a higher amount of messages per section is produced. The results for this experiment were inconclusive – no clear trend could be identified in the available simulation time.

In Figure 9 the accuracy results are shown. Surprisingly, the experiment showed that all penetration rates converge to a value between 87% and 93%, which would lead to the assumption, that the penetration rate has no substantial influence on the accuracy. Again, the specificity is the most influential indicator, showing the same pattern as the overall accuracy, while the sensitivity is only partially conclusive: it can be observed, that the higher penetration rates of 5%, 7% and 10% deliver better results than the lower rates. Though 5% and 7% perform better in the beginning, 10% has a steeper learning curve and seems to catch up in the end.

4.1.4. Section Length

The section length influences the amount of data that passes through the system, because smaller sections lead to a higher amount of mining instances. Furthermore, with smaller sections the real queue-end can be approximated to a higher degree if classified correctly, because the mean distance between the real queue-end and the forecasted queue-end (at the middle of the section) is smaller. Due to the lack of empirical data, we selected section lengths in a way, that results in a tolerable mean error of distance to the real queue-end (at most half of the section length). We experimented with 30 m, 50 m, 100 m, 150 m, and 300 m. We expected that a maximum in accuracy can be identified for a certain section length for the following reasons. Too small sections produce a too small amount of messages which are not sufficient to determine whether there is a queue-end or not. Too long sections are expected to be harmful to accuracy, because they can contain too many messages, which do not indicate the correct class. We

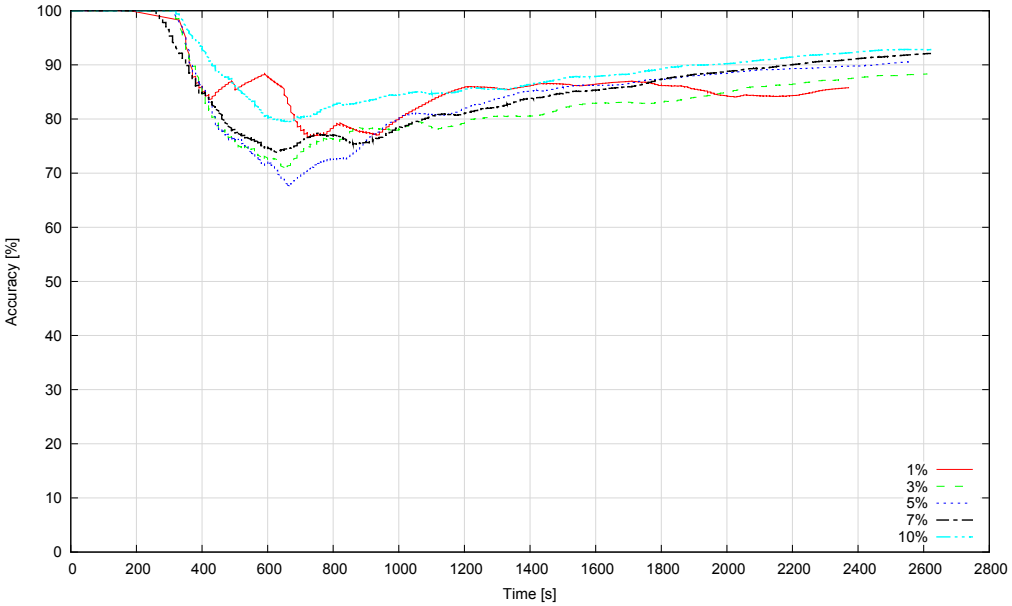


Figure 9: Accuracy for varying penetration rates

present the sensitivity results in Figure 10. In contrast to our expectations, it can be observed that for values between 30 m and 150 m there is no substantial difference neither in accuracy nor in specificity. However, sensitivity results get better with increasing section length. As a consequence, we should switch to a section size larger than our current default (100 m) when deploying the system and make additional experiments with higher section lengths to see if we reach a maximum.

4.1.5. Balancing

In the experiments before we identified the unbalanced ratio of examples for negatives and positives as one problem for getting good sensitivity values. This is a problem very well known in machine learning, but often not the only one which contributes to a bad learner performance [Batista et al. 2004]. We decided to analyze the problem and assess the influence of the unbalanced ratio of negative and positive examples for our three measures. We implemented a simple undersampling algorithm suited for streams. For each negative example we draw a random number from an equal distribution of numbers between 1 and 100. A negative example is dropped when the random integer is less or equal to the percentage of negatives which should be dropped (if 75% should be dropped all elements with integers below 76 are dropped). We used the default values for the parameters for all runs and tested percentages of elements between 0% and 90%. Figure 11 shows that dropping a high number of negatives is very beneficial for the sensitivity. But the results for the specificity reveal, that this has the opposite effect on the specificity (only reaches a little more than 50% for the 90% run), though the ratio of positives to negatives is almost equal ($1.2 : 1$). A good trade-off seems to be 85%, where the ratio is $0.85 : 1$. Here the sensitivity reaches values of a bout 65% while the specificity is around 82%. One approach to increase the overall accuracy could be to change the way we select the negative examples to be

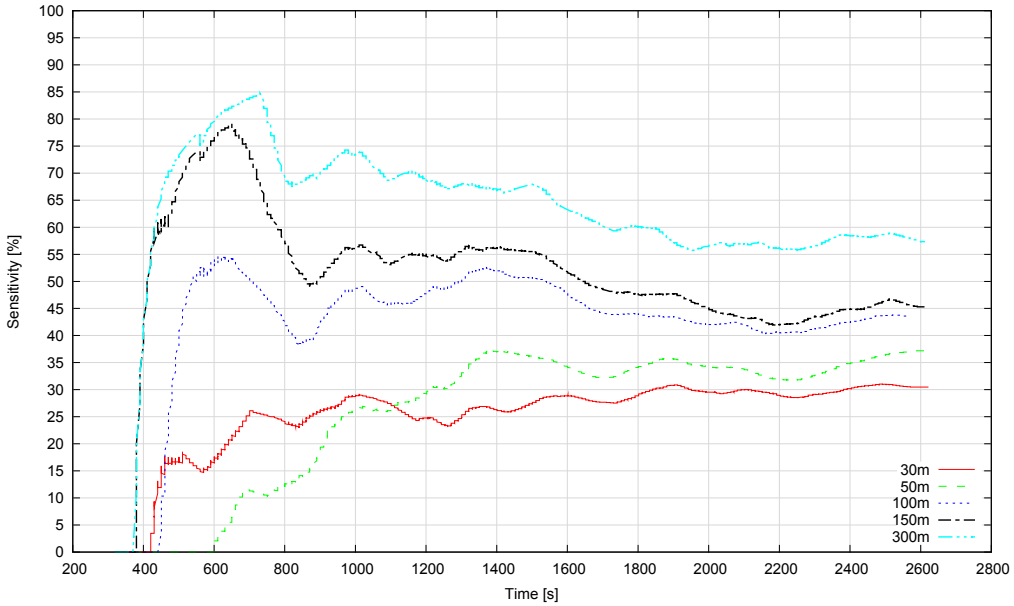


Figure 10: Sensitivity for varying section lengths

dropped. Other, more intelligent undersampling methods and also oversampling methods should be tested to see whether the type of sampling makes a difference. Batista et al. showed that the type of balancing technique can have an influence [Batista et al. 2004] for highly unbalanced data sets. Also a dynamic balancing would be interesting to investigate, because the ratio of positive to negative examples changes for instance with the queue-length (see Section 4.1.2).

4.2. Discussion of Results

In conclusion, the first results of the experiments show trends of influential parameters and reveal promising accuracy values also for smaller penetration rates. However, the results are not sufficient to draw final conclusions as we need to evaluate different traffic scenarios and fine tune the data stream mining algorithm. Especially the low sensitivity rates and inconclusive results need further investigation. Balancing positive and negative examples seems to be a promising direction for improving sensitivity, but it requires further investigation such that the overall accuracy is not decreased.

Nevertheless, it has been shown, that the framework can be used for the evaluation of traffic applications in a very flexible way. It can easily be extended to investigate other parameters or applications. The latter point is shown in the next section by applying the framework to a traffic state estimation scenario and evaluating the quality of data stream mining algorithms in this scenario.

5. Traffic State Estimation

The goal of this scenario is to determine the accuracy of data stream mining algorithms for the task of traffic state estimation. We want to vary not only different data stream mining

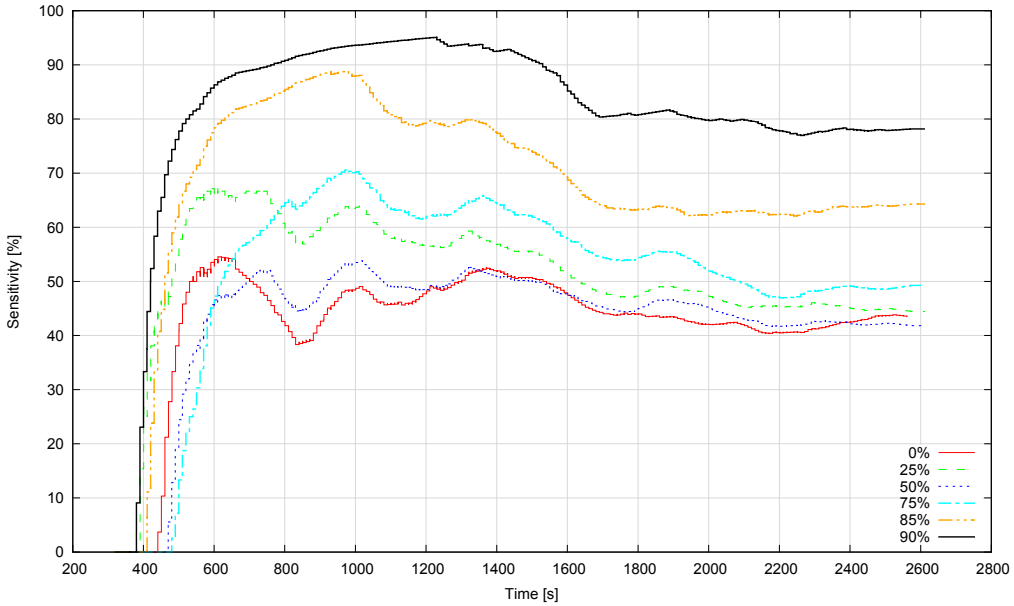


Figure 11: Sensitivity for varying undersampling rates for negative examples

algorithms to see which accuracies each of them can reach for the given task, but we also vary input attributes and input data. For example, it is very interesting to see if the inclusion of the VPD messages in the CoCar messages has an impact on accuracy.

In our approach, traffic state estimation is the determination of a class representing the current traffic state or *Level of Service (LOS)* for each section of a link. There have been several approaches for modeling these levels. For example, Kerner [2004] defined three states (free, synchronized and congested traffic), while the Highway Capacity Manual of the U.S. Transport Research Board proposes six states ranging from A to F (A = free flow, B = reasonably free flow, C = stable flow, D = approaching unstable flow, E = unstable flow, F = forced or breakdown flow). In this approach, we use a traffic state model, which has been described by the Federal German Highway Research Institute in [BASt 1999] and is widely implemented in German Traffic Management Centers. It distinguishes four levels of traffic: free, dense, slow-moving and congested traffic. We use these four levels as the classes for the data stream mining algorithms.

Furthermore, we utilize the definitions of the levels to generate ground truth class values in the traffic simulation. In [BASt 1999], for each level and a number of lanes a threshold or a range is defined for the mean speed and the local density at a stationary sensor. For example, for a highway with two lanes the traffic state is *slow-moving*, when the mean speed is above or equal 30 km/h and below 80 km/h and the density is below or equal 60 veh/km. In VISSIM it is possible to measure traffic data on a section basis, e.g., measuring the mean speed for all equal sized sections of a link. These values have been used to calculate the ground truth classes online and send them as ground truth messages to the DSMS. The sections in VISSIM comply with the sections in our spatial database.

For this scenario, we started with an artificial, but more complex road network than the one used in Section 4, depicted in Figure 12. The network consists of four links with two lanes each

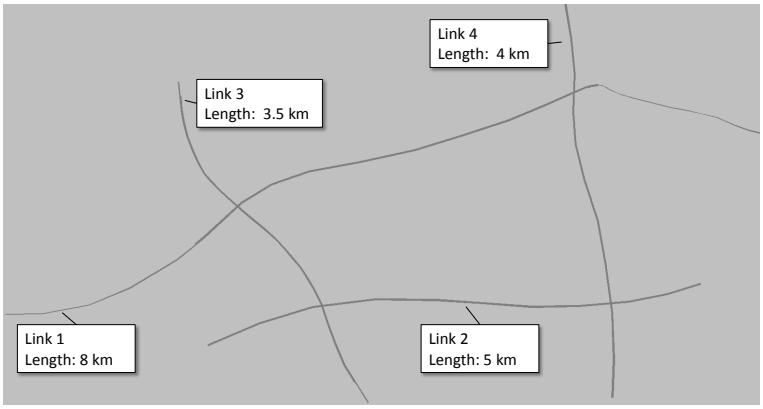


Figure 12: Artifical network for traffic state estimation

and covers an area of approximately 4 km by 8 km. For the links there is no speed limit defined by default. One difficulty when working with classification algorithms in traffic applications is, as we have seen from the previous case study, the balancing of training examples. There have to be sufficient examples for each of the traffic levels, such that all are well learned by the algorithm. This is challenging in simulation-based studies, as we have to prompt the simulation to produce these states in a very realistic manner. Congestion can be provoked in VISSIM by several means. One possibility is to use a stop line to block a lane.

This forces the vehicles to stop before this line, or if possible, change the lane, but leads to less realistic driver behavior (no zip merge). This can be avoided by using lane reduction, i.e., a link with two lanes is connected with a link with one lane, thereby also modeling a blocked lane. To create slow-moving traffic, reduced speed areas (a defined part of a link with a speed limit) in combination with varying traffic volumes can be utilized.

In the DSMS, the CoCar messages and ground truth messages have been integrated and aggregated using a 60 s window which slides every 10 s. All other steps are the same as in the queue-end detection scenario, except that the Map Matching of the ground truth messages is obsolete as they already contain the section and link identifiers. For the data stream mining evaluation in this scenario, we used the prequential method described in Section 2 with a window size of 20 elements.

5.1. Evaluation and Results

The goal in this scenario is to evaluate how different data stream mining algorithms perform for traffic state estimation using traffic information from CoCars. Also various parameters, which are assumed to have an influence on the mining accuracy, have been investigated. To rate the accuracy for the data stream mining, we use the windowed Interleaved Test-Then-Train evaluation method described in Section 2.3. We record the number of combinations of correct and predicted classes of the classification result in a confusion matrix which is shown in Table 1.

The columns are the predicted classes and the rows the correct classes. The values a_{ii} on the diagonal represent the number of correctly identified traffic states for each class. All other cells represent the number of elements which have been falsely predicted to class `p<Class>`, but should be `c<Class>`. In the example table, a_{12} is the number of elements which have been predicted with class `dense`, which were actually class `free`. With the recorded values we can

	pFree	pDense	pSlow	pCongested
cFree	a_{11}	a_{12}	a_{13}	a_{14}
cDense	a_{21}	a_{22}	a_{23}	a_{24}
cSlow	a_{31}	a_{32}	a_{33}	a_{34}
cCongested	a_{41}	a_{42}	a_{43}	a_{44}

Table 1: Confusion matrix for traffic states

now calculate the mining accuracy. For each class the number of correctly classified elements in this class is divided by the overall number of classified elements. To reward, when the algorithm was only slightly wrong, but not completely wrong, we introduce a constant factor 0.6 with which we multiply the count of similar traffic state classes and add the result to the number of correctly classified elements. The resulting equations for the accuracy in each class are defined as follows:

$$\begin{aligned} \mathit{FreeAcc} &= \frac{a_{11} + (0.6 \cdot a_{21})}{a_{11} + a_{21} + a_{31} + a_{41}} \\ \mathit{DenseAcc} &= \frac{a_{22} + (0.6 \cdot (a_{12} + a_{32}))}{a_{12} + a_{22} + a_{32} + a_{42}} \\ \mathit{SlowAcc} &= \frac{a_{33} + (0.6 \cdot (a_{23} + a_{43}))}{a_{13} + a_{23} + a_{33} + a_{43}} \\ \mathit{CongestedAcc} &= \frac{a_{44} + (0.6 \cdot a_{34})}{a_{14} + a_{24} + a_{34} + a_{44}} \end{aligned}$$

The overall accuracy is calculated by:

$$\mathit{Acc} = \frac{(a_{11} + (0.6 \cdot a_{21})) + (a_{22} + (0.6 \cdot (a_{12} + a_{32}))) + (a_{33} + (0.6 \cdot (a_{23} + a_{43}))) + (a_{44} + (0.6 \cdot a_{34}))}{\sum_{i,j=1}^{n} a_{ij}}$$

The evaluation process comprised several simulation runs with varying parameters for the traffic simulation and the data stream mining algorithms. Each run started with a 420 s setup phase in which the network was populated with traffic. Afterwards, in the evaluation phase of 1800 s the data stream processing and mining was carried out to analyze the traffic state estimation accuracy. For each run a classifier was learned from scratch. If not varied in a test, we used for the segment length 400 m and 10% CoCar penetration rate as default values. The concept-adapting Hoeffding Option Tree with Naïve Bayes prediction strategy was used as the default classifier as it delivered the best results in comparison to other algorithms.

5.1.1. Inclusion of Probe Data

In the first tests we made, we only used EBL and WLA messages and the corresponding data for aggregation and mining. As these messages are only created in situations with congested traffic, the class `congested` has been classified in these tests in the mean with an accuracy over 96% while other classes had a very bad accuracy. Therefore, the simulation was extended by introducing an additional CoCar message type, the Vehicle Probe Data (VPD) message type. These messages are sent by the vehicles periodically, i.e., every 10 s, and only contain basic information such as position, speed, and acceleration. This enabled the mining algorithms to learn the

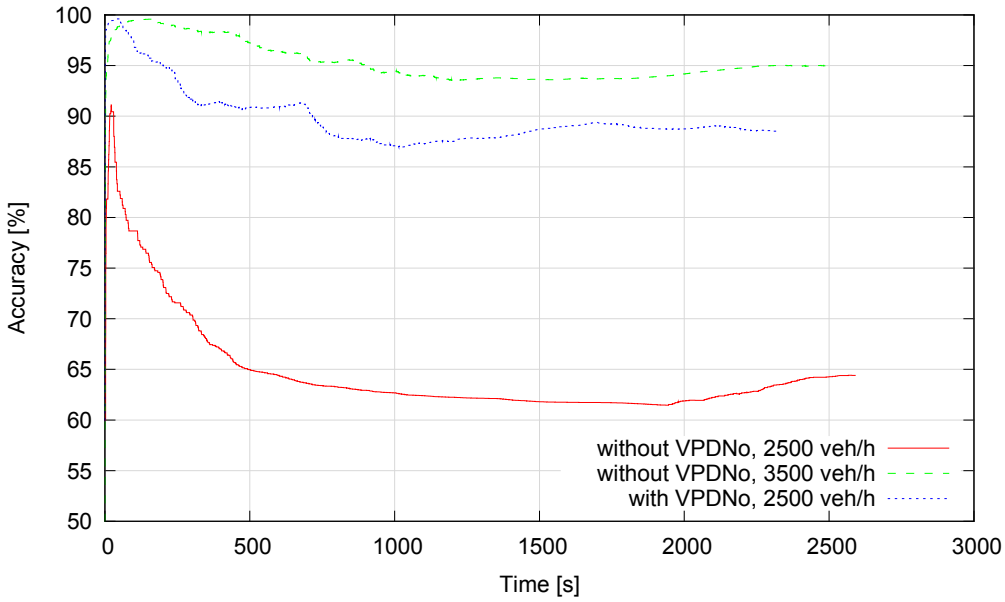


Figure 13: Accuracy for dense traffic with varying traffic volume and with and without VPD message count as attribute

other classes with a higher accuracy. Furthermore, the preliminary traffic volume of 2500 veh/h turned out not to be sufficient to learn dense traffic very well, but only free flow. This was due to the fact, that with 2500 veh/h only at the location where the link was narrowed congested traffic was created and the traffic was flowing freely otherwise. To produce also situations with dense and slow-moving traffic, the traffic volume was increased to 3500 veh/h. However, this led to false classification of many elements with class `dense` to class `free` and the few examples for the class `free` were not sufficient to learn this class with an acceptable accuracy. Therefore, different traffic volumes were used on different links to balance the number of examples for each class. Still, the algorithms were not able to distinguish properly between the classes `dense` and `free`. As the ground truth also includes the density as a distinctive property for the different traffic states, we added the number of VPD messages as a new input attribute to the training examples. This led to an increase in accuracy for the classes `free`, `dense` as well as `slow-moving` as can be seen in Figure 13 for dense traffic. The diagram also shows the increase in accuracy between 2500 veh/h and 3500 veh/h. Hence, in further experiments we used VPD messages and the VPD message count as an attribute.

5.1.2. Section Length

As seen from the queue-end detection case study, the section length can have an impact on the mining accuracy. Therefore, we also experimented with varying section lengths in this scenario. Section lengths between 100 m and 1000 m have been tested, geared to the distances between inductive loop detectors on German highways. It turned out that longer section lengths are less beneficial for the accuracy as depicted in Figure 14. While lengths between 800 m and 1000 m produce results of around 80%, lengths of 200 m and 400 m deliver the best results of approximately 87%. This is due to the fact, that in longer sections the probability is higher that

they include more than one traffic state class. Thus, a section length of 400 m is used as a default parameter in further tests. These results may also depend on the traffic volume in the network and therefore, further tests with varying traffic volumes and section lengths have to be carried out.

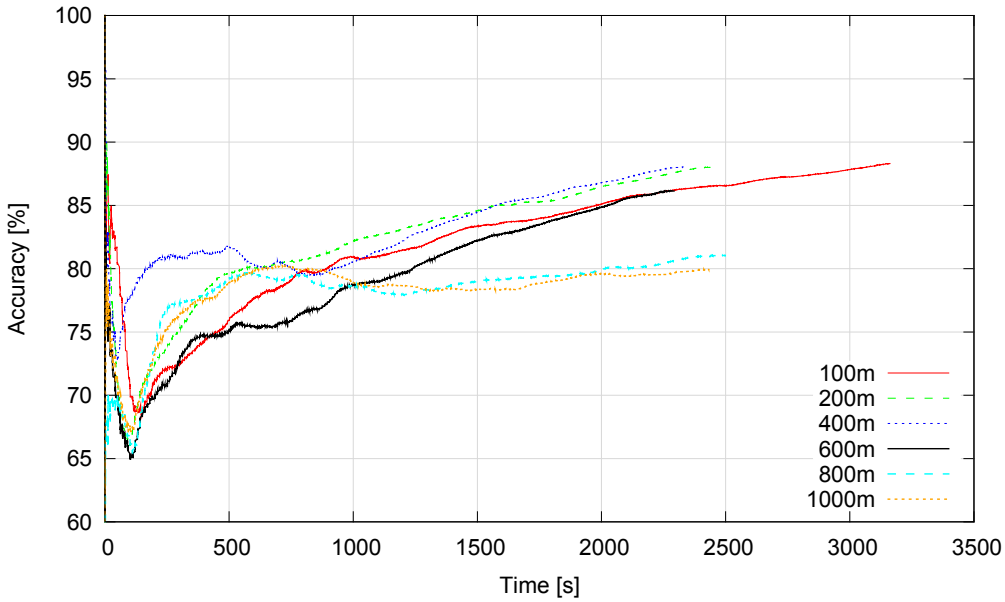


Figure 14: Accuracy for varying section lengths

5.1.3. CoCar Penetration Rate

It seems to be plausible that, with an increasing penetration rate, more information is available in each section and therefore, the mining accuracy should be higher. In the results of the queue-end detection case study, the CoCar penetration rate turned out not to have too much impact on the mining accuracy. To see if this is also the case for the traffic state estimation, we tested CoCar penetration rates between 5% and 20%. Figure 15 indicates that the above assumption has also been proven wrong for traffic state estimation. The results vary slightly between 88% and 91%, but no clear trend can be identified. However, an increased penetration rate inevitably leads to a higher network coverage and hence, is always rated as beneficial.

5.1.4. Comparison of Data Stream Mining Algorithms

In the MOA mining framework various stream mining algorithms are readily available, most of them based on the basic algorithm of Hoeffding Trees detailed in Section 2.3. To find out, how well the algorithms are suited to be used in the application of traffic state estimation, several tests with varying algorithms have been made. One distinction of the implementations of the algorithms is in their prediction strategy. The prediction strategy is the method to decide at the leaves in the decision tree, which class for the element to classify is assigned. There are three different prediction strategies implemented in MOA: Majority Voting, Naïve Bayes Leaves and Adaptive Hybrid [Bifet and Kirkby 2009a]. While majority voting assigns the class, for which

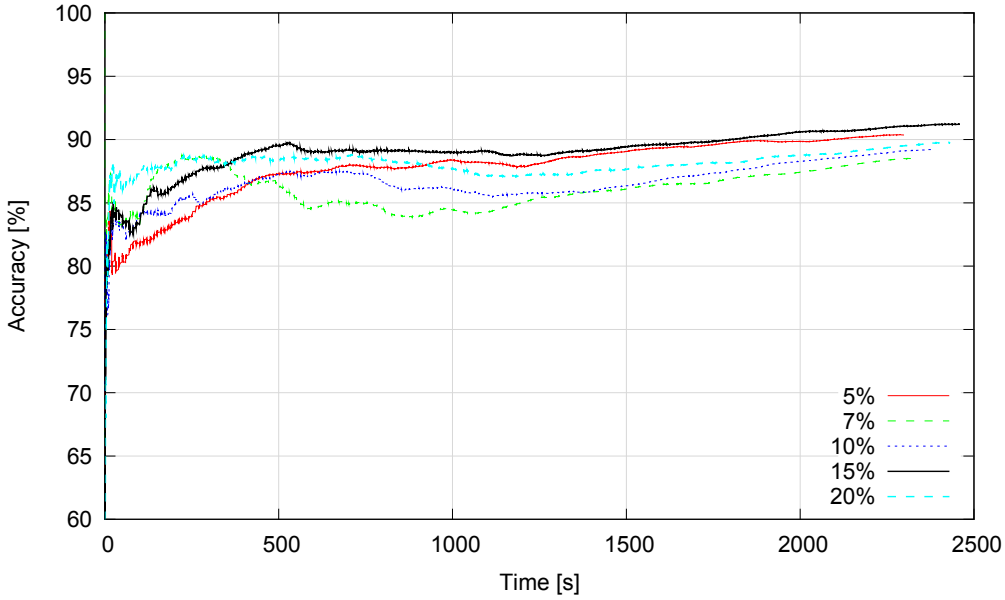


Figure 15: Accuracy for varying CoCar penetration rates

the most examples arrived at this leaf, the Naïve Bayes Leaves uses the Bayes theorem and the probabilities of the attribute values to determine the probability of each class and selects the class with the highest probability. The Adaptive Hybrid strategy is a combination of both. For each strategy the accuracy is calculated and the Naïve Bayes decision is only used if its accuracy was higher before. In Figure 16 the accuracy results of all three strategies for a Hoeffding Tree are shown. On average, the Naïve Bayes and the Adaptive Hybrid strategy were very similar in their results, but both outperformed the Majority Voting strategy.

Ensemble algorithms learn and use multiple models in parallel for classifying an element using a basic model type (the *base learner*), e.g., a Hoeffding Tree. In addition, each example in the training set is weighted to determine its importance and the different models can also be weighted, to denote how reliable their classification results are. Algorithms differ in how they weight the examples and the models [Bifet and Kirkby 2009a]. The most famous variants are *bagging* and *boosting* algorithms. Bagging algorithms train their different models on different training sets which have been assembled by randomly drawing (with replacement) a set of examples from a common training set with the same size as this training set. The votes of all models have the same weight, i.e., they are unweighted. Boosting algorithms combine several so called weak learners (very simple models) to build a strong learner. This is done in an iterative process, where models are chained and successive models learn from the failures of previous ones. The models are weighted according to their overall accuracy, which represents their reliability. Implementations of various ensemble algorithms are included in MOA and we analyzed all of them in our evaluation framework using the Hoeffding Tree as the base learner. The ensemble algorithms comprise the online boosting and bagging algorithms by Oza and Russell [Oza and Russell 2001, Oza 2006], the Online Coordinate Boosting algorithm presented in [Pelossof et al. 2010] and an Option Tree algorithm introduced in [Bifet and Kirkby 2009a]. Option trees grow

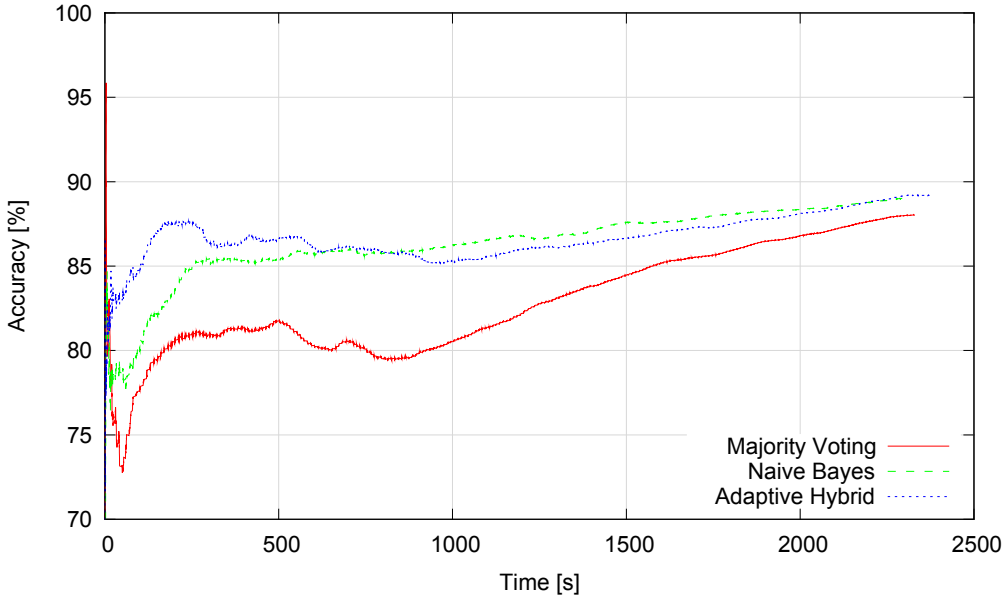


Figure 16: Accuracy for Hoeffding Trees with varying prediction strategy

new alternative branches at so called option nodes. Examples are pipelined to all branches and leaves connected with the option node and the overall result is determined by combining the results of the nodes beneath the option node. We do not go into the details of these algorithms, but refer the interested reader to the cited papers. Figure 17 shows the results for the tested ensemble algorithms. The boosting algorithm is slightly better than the other algorithms with accuracies of up to 92%. On average, it delivers also only a slightly better accuracy than the best single classifier algorithm in Figure 16.

Finally, also algorithms adapting to concept drift have been analyzed according to their accuracy. The scenario does not exhibit a concept drift. Nonetheless, we wanted to see, if the concept-adapting algorithms perform comparably well to other algorithms. Our goal was not to assess the capability of these algorithms to actually react to a concept drift. This could be subject to future work. MOA provides four different algorithms. They implement the online bagging algorithm by Oza and Russell using an Adaptive-Size Hoeffding Tree (ASHT) as the base learner. The ASHT is limited in the number of splitting nodes. When the maximum number is reached, nodes are deleted. [Bifet and Kirkby 2009a] argue, that smaller trees can adapt faster to changes than bigger trees, while the latter ones are more accurate for stationary streams due to the higher number of examples seen. Furthermore, they combine the Oza and Russell bagging with a concept detection algorithm called ADWIN, which has been proposed in [Bifet and Gavalda 2007]. ADWIN maintains a window which can adapt its size to the degree of change in the stream, meaning that the window always represents the average of the streams numeric attributes. If the current window content does not represent the average anymore, concept drift has taken place and the window adapts its size. Furthermore, an implementation of a single classifier combined with a concept drift detection method from [Gama et al. 2004] has been integrated as well as an adaptive version of the Hoeffding Option Tree. Details about all implemented al-

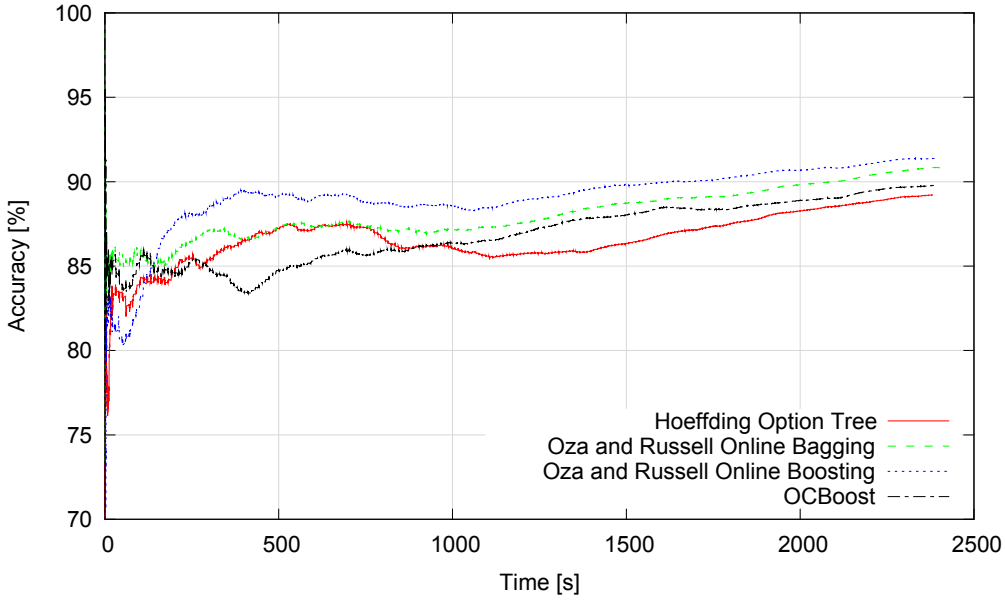


Figure 17: Accuracy of multiple ensemble algorithms

gorithms in MOA can be found in [Bifet and Kirkby 2009b]. Figure 18 reveals that all of the concept-adapting algorithms perform with almost identical accuracy. Compared to the ensemble techniques in Figure 17 they do not provide substantial improvements in accuracy – actually, the Oza and Russell online boosting is better in the average.

5.1.5. Multiple Runs

So far, a new classification model has been used for each simulation run. Each simulation run was 1800 s long. If the model is trained over a longer period of time, it is assumed that the accuracy will also be increased, as the model has more examples to learn on. Therefore, multiple simulation runs have been carried out, reusing the trained model in each run and thereby incrementally extending it. The results shown in Figure 5.1.5 corroborated our assumption. The overall accuracy of the mining increased from approximately 88% to 91%, which denotes an improvement, but not as high as expected.

5.1.6. Applicability to Realistic Maps

Using artificial maps for the training of a model is beneficial, as traffic situations can be controlled to create sufficient examples for every kind of traffic state class. However, these networks may not reflect traffic situations as they would occur in a realistic road network. Hence, we rebuild a German highway interchange nearby Mönchengladbach, where the highways A52 and A61 are crossing. The network covers a 3 km by 3 km area and includes all links, slip roads and exits, depicted in Figure 20(a) with the corresponding OpenStreetMap section[11] in Figure 20(b).

[11] `http://www.openstreetmap.org`

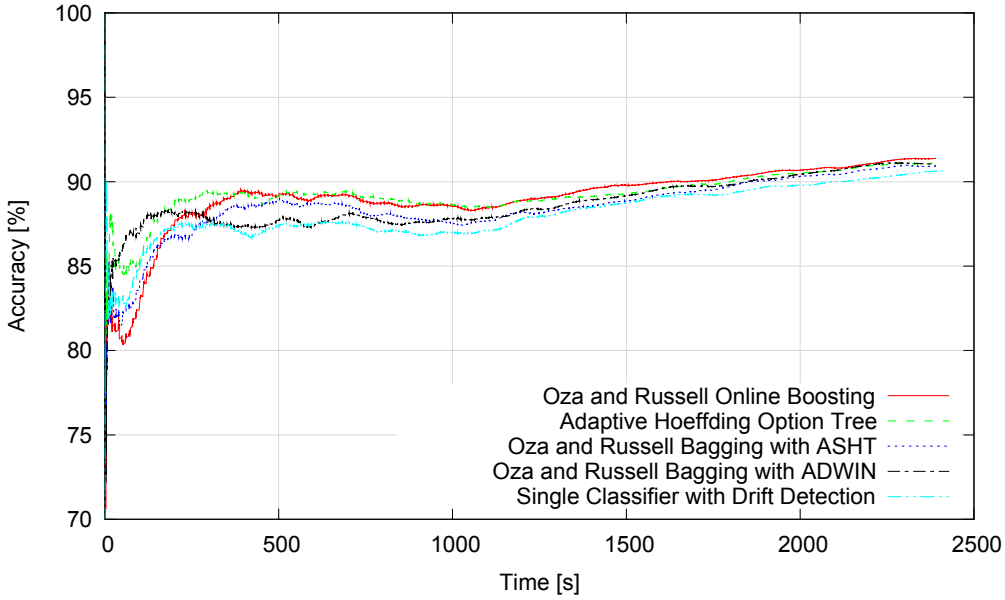


Figure 18: Accuracy of multiple concept-adapting algorithms

In the simulation runs the traffic volume of each link was varied to simulate realistic traffic situations under varying loads in the network. We tested traffic volumes of 2000 veh/h to 5000 veh/h, which were the same for all links in the same run. For the tests, a trained model with very good accuracies in the tests with the artificial network was used. The model was only used for classification and was not further trained during these runs. In Figure 21 the results show, that for a very low traffic volume of 2000 veh/h the model can estimate the traffic state with a very good (98%) accuracy. A very high traffic volume of 5000 veh/h reaches an acceptable value of (87%) accuracy. 3000 veh/h and 4000 veh/h reach 83% to 84%. This can be explained by a higher rate of dense and slow-moving traffic occurring in these simulation runs. The accuracy for these classes was also lower in the artificial network, as it is easier to predict the 'extreme' situations (free and congested traffic) than to predict dense and slow traffic.

5.2. Discussion of Results

As in the case study of queue-end detection, for traffic state estimation we also observed that a balanced training data set is important for training a classifier. Therefore, the artificially road network is constructed in such a way that training data for the four traffic classes are generated almost equally. The last experiment has shown that the classifier trained on the artificial network delivers also very good results on a real road network.

In general, the results for traffic state estimation are more conclusive than for queue-end detection. We were able to identify a turning point for the section length (400 m) and higher traffic volumes generate a higher accuracy. In addition, if the classifier is trained in multiple runs, we can see that after the second run, the classifier is very stable and the accuracy changes only very little at a level of 90%. The comparison of different data stream mining algorithms

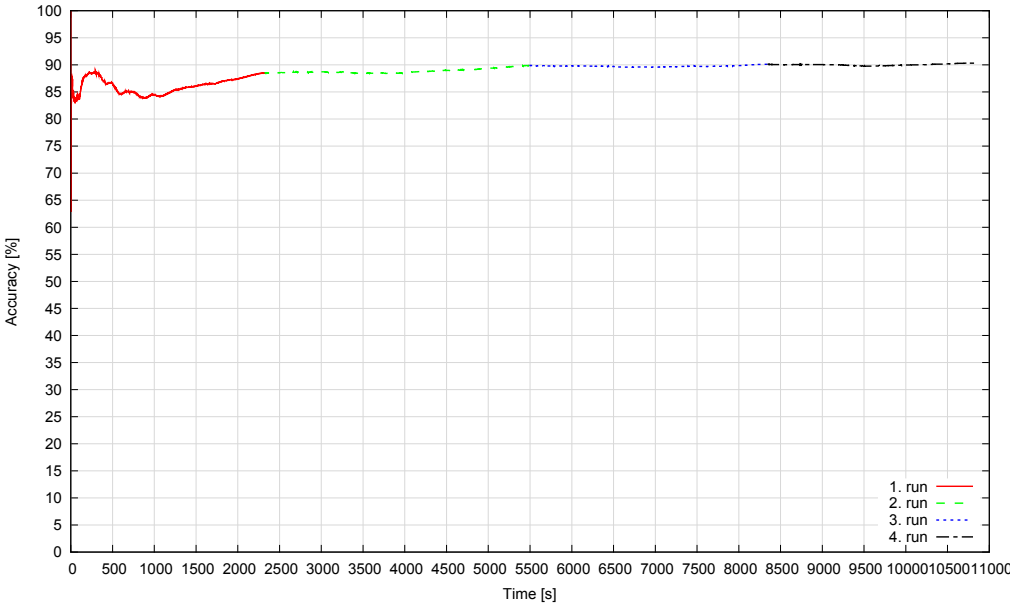


Figure 19: Development of accuracy in successive runs

gave also similar results as the ICDM contest for data mining in traffic applications [Wojnarski et al. 2010], in which ensemble methods performed best.

6. Related Work

Data stream management systems and data stream mining is a field of research that has received increasing attention in the last years. It also found its way into traffic applications in the recent years. For example, the product MineFleet [Kargupta et al. 2010] integrates data stream mining into vehicle embedded systems for fleet monitoring to analyze vehicle health, emissions, or driver behavior. Another approach detecting driver behavior is presented by Horovitz et al. [Horovitz et al. 2007], which uses a combined approach of unsupervised data stream clustering and fuzzy logic to detect drunken driver behavior. Liu et al. [Liu et al. 2006] propose a distributed traffic stream mining system for the determination of congestion level using Frequent Episode Mining. On a central server frequent patterns are determined based on historical data and are then distributed to stationary detectors, which use the pattern to classify data from the sensors. The Linear Road benchmark for DSMS [Arasu et al. 2004] allows for performance measurements (throughput and response time) and comparison of multiple DSMS. The benchmark is based on a simulated traffic scenario which calculates tolls for vehicles driving into areas with congestion and accidents. The traffic scenario is a means to produce data for system stress tests, but does not aim at analyzing the utilized traffic application. Hence, complex tasks, such as mining, are not analyzed. Also global players in data management work on solutions for traffic applications based on data stream management. Recently, IBM proposed a real-time traffic information management system based on its streaming platform Infosphere Streams [Biem et al.

(a) Road network in VISSIM (b) Real network in OpenStreetMap

Figure 20: Interchange Mönchengladbach

2010]. They used GPS data from taxis and trucks of the city of Stockholm to calculate travel times and shortest paths between city parts to demonstrate the effectiveness of their system, but did not utilize any data stream mining techniques. Microsoft integrates handling of spatial data with data stream capabilities implemented in StreamInsight [Ali et al. 2010b]. Furthermore, they demonstrated the usefulness of their platform by an application monitoring the shuttles on the Microsoft campus and analyzing and balancing their utilization [Ali et al. 2010a].

For mobile detection of traffic data, there have been several studies, which aim at rating the accuracy of the created traffic information. Especially, in the field of anonymously collected Floating Phone Data the interest is high. These studies can be divided mainly into field studies and simulation studies. Intensive simulation studies have been carried out by Fontaine and Smith [2005; 2007]. They investigated the influence of different parameters in the road network and mobile network on the accuracy of traffic information, such as link speed. However, they do not consider the characteristics of the processing information system and do not incorporate analysis techniques, such as data mining, to derive events or traffic information from the collected data. In our work, we investigate the influence of varying road and traffic parameters, and we also take into account the special features of a data stream management system, such as window size, sliding step, and of data stream mining, such as concept drift.

For the first scenario, the investigation of queue-end detection algorithms is another important line of work. Huber [2001] studied the opportunities in traffic information collection using XFCD. He analyzed, which in-vehicle information, e.g., collected over CAN bus or sensors, can be used to detect certain local traffic events. One example he investigated is the queue-end detection using a deceleration parameter, an indicator for low speed level, right turn signals, the road type, and warning flashers. The event determination is implemented using fuzzification and a rule base. To reach 90% probability that a vehicle arrives at an incident in a one minute interval a penetration rate of 6.9% is required already. In contrast, our system would be able to announce a queue-end in a much lower time interval (10 s) and is therefore more suitable for local hazard warnings. Furthermore, the system does not use any learning. Fuzzification and rules are manually defined. The use of fuzzy rules would be an interesting extension to our method. Chan et al. [2003] present a system for real-time queue tracking based on the average speed detected on road sections by identifying three traffic zones and analysing the arrangement of these zones. Though, the authors base their work on stationary detection and have to rely on larger sections (minimum of 500 m between loop detectors on a comprehensively equipped highway), the idea of identify-

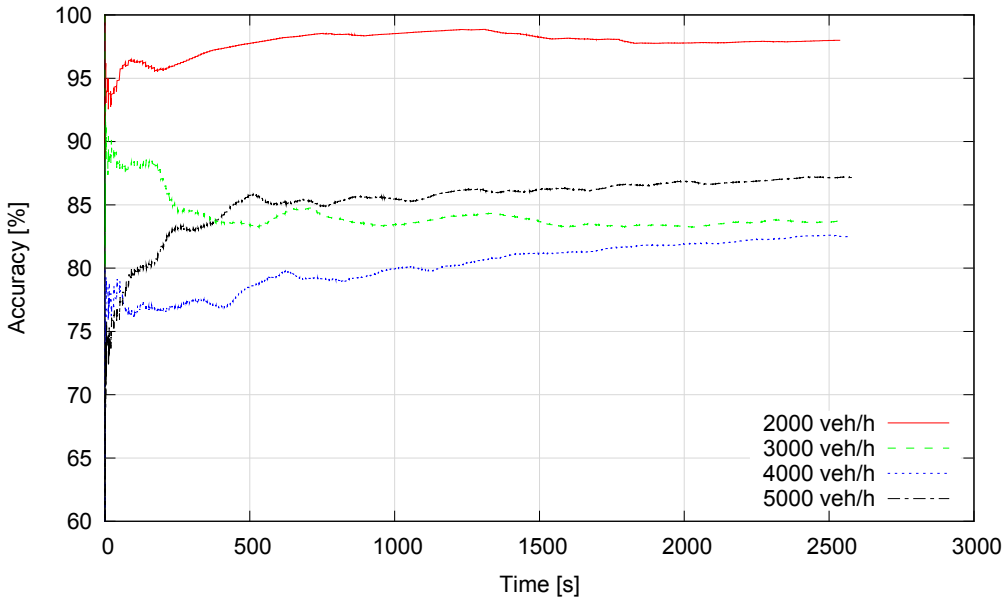


Figure 21: Accuracy using a trained classifier on a realistic map for simulation

ing the parts of the queue seems appealing and could be integrated in our system to enhance the quality of the detection mechanism. More current work on queue-end detection uses methods from the field of artificial intelligence. Khan [2007] presents a simulation study also based on VISSIM using stationary sensors as data source. They analyze the data using Artificial Neural Network models, predicting the current queue length based on accumulated numbers of cars and trucks at fixed locations. Although they postulate real-time processing in their information system, the used algorithm is not capable of online learning, i.e., it cannot adapt to concept changes, which can be achieved by using data stream mining algorithms as proposed in our approach.

Well-known techniques for traffic state estimation and forecasting comprise the use of Kalman filters [Wang and Papageorgiou 2005, Ben-Akiva et al. 2001], time-series analysis, e.g., ARIMA or state space models in [Stathopoulos and Karlaftis 2003], or model-based approaches [Kerner 2004, Nagel and Schreckenberg 1992]. One of the main goals of the evaluation framework was to investigate the applicability of data stream mining algorithms to traffic applications. Therefore, we do not consider the former approaches. In traffic state estimation and forecasting, neural networks have been used in many areas of transportation, because they are well suited to model non-linear processes [Bogenberger et al. 2003]. They are mainly used in short-term prediction for traffic volumes or travel times. For example in [Chen and Grant-Muller 2001] a dynamic neural network approach based on a resource allocating network has been proposed. An interesting hybrid approach for the prediction of average speeds is presented in [Bogenberger et al. 2003]. They combine a linear and a non-linear model. The linear model uses an ARIMA algorithm for stationary traffic situations while neural networks are used as a non-linear model for unstable, non-stationary traffic situations, e.g., congested traffic. The approach predicts average speeds using speed and traffic volume measurements from stationary detectors. As pointed out in [Gama and Rodrigues 2009] neural networks can easily be used in data stream processing. Hence, it

would be interesting to integrate also neural network algorithms into the stream mining framework we used and compare its performance with the concept-adapting ensemble algorithms.

7. Conclusion

Car-to-X communication enables the collection of sensor data generated in cars and the exploitation of this data for traffic information and management applications. This scenario is a challenging one for data management solutions, as huge amounts of data have to be processed and analyzed in real-time. What is more, the desire to integrate data from multiple sources poses further demands on the functionality of the data management system. For example, additional data sources such as weather information and historical data could improve the quality of the generated traffic data. Data stream management systems and data stream mining algorithms appear to be appropriate solutions for some of the problems above. Still, many questions (e.g., Which data sources should be used? Which mining algorithms produce good results? Which level of data quality can be achieved? What are good parameters for the DSMS and the traffic applications at hand?) remain open.

To help in finding answers to these questions, we have developed a framework for simulation-based evaluation of traffic applications. The framework consists of a traffic simulation, a data stream management system, data stream mining algorithms, and a spatial database system. It enables us to evaluate the influence of parameters of road networks and traffic scenarios as well as the impact of characteristics of the information system and the data mining algorithms. In two case studies, namely queue-end detection and traffic state estimation, we illustrated that our approach yields feasible means to investigate effects of these parameters.

The results of the evaluation carried out show that trends in accuracy and specificity based on varying parameter configurations can be observed already with small sets of training instances. For traffic state estimation, it is important that cars periodically send vehicle probe data as otherwise only free or congested traffic can be accurately detected. While traffic volume is a strong factor influencing data mining accuracy in both case studies, the penetration rate showed no crucial effect. The section length had a high impact on the accuracy as well, but the queue-end detection performed best with a section length of less than 100 m, whereas traffic state estimation had better results with 200 m or 400 m section length. For the data management system, we studied the effect of different window sizes and also analyzed the suitability of data stream mining algorithms for traffic state estimation. Ensemble methods seem to achieve the best performance. Overall, the flexibility of our framework enables the analysis of the effect of a variety of parameters in the traffic scenario, the data management system, and the data stream mining,

The two case studies presented yielded interesting results already, further evaluations and assessment of specific issues left open are the consequent next step in our work plan. We will investigate means to enhance the accuracy and sensitivity in queue-end detection and traffic state estimation, for example, by enhancing Map Matching accuracy, by giving more weight to certain input attributes, or by including data of adjacent sections. Also, taking into account other data attributes and data sources for the mining algorithms is easily possible in our data fusion architecture. A further extension would be the use of historical data in queries, which is also possible with our architecture. Application data can be stored in a database and used, for instance, to do a short-term traffic state forecast or enhance the quality of the traffic state estimation. In conclusion, it has been shown that our framework is a powerful instrument to study traffic applications and the corresponding data management system.

8. Acknowledgments

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under the grant 01BU0915 (Project Cooperative Cars eXtended, `http://www.aktiv-online.org/english/aktiv-cocar.html`) and by the Research Cluster on Ultra High-Speed Mobile Information and Communication UMIC (`http://www.umic.rwth-aachen.de`). We thank the PTV AG for kindly providing us with a VISSIM license. We also thank the reviewers for their valuable comments.

References

Abadi, D.J., Ahmad, Y., Balazinska, M., Çetintemel, U., Cherniack, M., Hwang, J.H., Lindner, W., Maskey, A., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.B., 2005. The Design of the Borealis Stream Processing Engine, in: Proc. 2nd Biennal Conference on Innovative Data Systems Research (CIDR), Asilomar, CA, USA. pp. 277–289.

Aberer, K., Hauswirth, M., Salehi, A., 2006. A Middleware for Fast and Flexible Sensor Network Deployment, in: Dayal, U., Whang, K.Y., Lomet, D.B., Alonso, G., Lohman, G.M., Kersten, M.L., Cha, S.K., Kim, Y.K. (Eds.), Proc. 32nd Intl. Conference on Very Large Data Bases (VLDB), ACM Press. pp. 1199–1202.

Aggarwal, C., Yu, P., 2008. A Framework for Clustering Uncertain Data Streams, in: IEEE 24th International Conference on Data Engineering, IEEE. pp. 150–159.

Ahmad, Y., Çetintemel, U., 2009. Data Stream Management Architectures and Prototypes, in: Liu and Özsu [2009]. chapter D. pp. 639–643.

Ali, M., Chandramouli, B., Raman, B.S., Katibah, E., 2010a. Real-Time Spatio-Temporal Analystics using Microsoft StreamInsight, in: Proc. of the 18th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, San Jose, CA, USA. pp. 542–543.

Ali, M.H., Chandramouli, B., Raman, B.S., Katibah, E., 2010b. Spatio-Temporal Stream Processing in Microsoft StreamInsight. IEEE Data Engineering Bulletin 33, 69–74.

Angelov, P., Zhou, X., 2008. Online Learning Fuzzy Rule-based System Structure from Data Streams, in: Proc. of the IEEE International Conference on Fuzzy Systems, pp. 915–922.

Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J., 2003. STREAM: The Stanford Stream Data Manager, in: Halevy, A.Y., Ives, Z.G., Doan, A. (Eds.), Proc. of the ACM SIGMOD Intl. Conference on Management of Data, ACM, San Diego, California, USA. p. 665.

Arasu, A., Babu, S., Widom, J., 2006. The CQL Continuous Query Language: Semantic Foundations and Query Execution. The VLDB Journal 15, 121–142.

Arasu, A., Cherniack, M., Galvez, E., Maier, D., Maskey, A., Ryvkina, E., Stonebraker, M., Tibbetts, R., 2004. Linear Road: A Stream Data Management Benchmark, in: Proc. of the 30th International Conference on Very Large Data Bases (VLDB), VLDB Endowment. pp. 480–491.

Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J., 2002. Models and Issues in Data Stream Systems, in: Popa, L. (Ed.), Proc. 21st ACM Symposium on Principles of Database Systems (PODS), ACM Press, Madison, Wisconsin. pp. 1–16.

Bai, Y., Thakkar, H., Luo, C., Wang, H., Zaniolo, C., 2006. A Data Stream Language and System Designed for Power and Extensibility, in: Yu, P.S., Tsotras, V.J., Fox, E.A., Liu, B. (Eds.), Proc. ACM Intl. Conf. on Information and Knowledge Management (CIKM), Arlington, Virginia. pp. 337–346.

BASt, 1999. Merkblatt für die Ausstattung von Verkehrsrechnerzentralen und Unterzentralen (MARZ). Bundesanstalt für Straßenwesen. (in German).

Batista, G., Prati, R., Monard, M., 2004. A Study of the Behavior of Several Methods for Balancing Machine Learning Training Data. ACM SIGKDD Explorations Newsletter 6, 20–29.

Ben-Akiva, M., Bierlaire, M., Burton, D., Koutsopoulos, H., Mishalani, R., 2001. Network State Estimation and Prediction for Real-time Traffic Management. Networks and Spatial Economics 1, 293–318.

Biem, A., Bouillet, E., Feng, H., Ranganathan, A., Riabov, A., Verscheure, O., Koutsopoulos, H., Moran, C., 2010. IBM InfoSphere Streams for Scalable, Real-Time, Intelligent Transportation Services, in: Proc. of SIGMOD'10, pp. 1093–1104.

Bifet, A., Gavalda, R., 2007. Learning from Time-changing Data with Adaptive Windowing, in: Proc. of the SIAM International Conference on Data Mining, pp. 443–448.

Bifet, A., Holmes, G., Kirkby, R., Pfahringer, B., 2010a. MOA: Massive Online Analysis. Journal of Machine Learning Research 11, 1601–1604.

Bifet, A., Holmes, G., Pfahringer, B., Kranen, P., Kremer, H., Jansen, T., Seidl, T., 2010b. MOA: massive online analysis, a framework for stream classification and clustering, in: Proc. of the Intl. Workshop on Handling Concept Drift in Adaptive Information Systems (HaCDAIS), pp. 3–16.

Bifet, A., Kirkby, R., 2009a. Data Stream Mining - A Practical Approach. University of Waikato. `http://moa.cs.waikato.ac.nz/wp-content/uploads/2010/05/StreamMining.pdf`.

Bifet, A., Kirkby, R., 2009b. Massive Online Analysis - Manual. The University of Waikato. `http://moa.cs.waikato.ac.nz/wp-content/uploads/2010/05/Manual.pdf`.

Bogenberger, K., Belzner, H., Kates, R., 2003. Ein hybrides Modell basierend auf einem Neuronalen Netz und einem ARIMA-Zeitreihenmodell zur Prognose lokaler Verkehrskenngrössen. Strassenverkehrstechnik 47, 5–12. (in German).

Chan, T.N., Lee, C., Hellinga, B., 2003. Real-time Identification and Tracking of Traffic Queues Based on Average Link Speed, in: 82nd Annual Meeting of the Transportation Research Board.

Chen, H., Grant-Muller, S., 2001. Use of Sequential Learning for Short-term Traffic Flow Forecasting. Transportation Research Part C: Emerging Technologies 9, 319 – 336.

Cheng, J., Ke, Y., Ng, W., 2008. A Survey on Algorithms for Mining Frequent Itemsets over Data Streams. Knowledge and Information Systems 16, 1–27.

Cherniack, M., Zdonik, S., 2009. Stream-Oriented Query Languages and Architectures, in: Liu and Özsu [2009]. chapter S. pp. 2848–2854.

Delot, T., Ilarri, S., Thilliez, M., Vargas-Solar, G., Bellengier, M., 2010. Highly Mobile Query Processing, in: International Conference on Ambient Systems, Networks and Technologies (ANT 2010), Paris (France), Austrian Computer Society (OCG). pp. 101–108.

Domingos, P., Hulten, G., 2000. Mining High-speed Data Streams, in: Proc. of 6th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, ACM. pp. 71–80.

Domingos, P., Hulten, G., 2003. A General Framework for Mining Massive Data Streams. Journal of Computational and Graphical Statistics 12, 945–949.

Etzion, O., Niblett, P., 2011. Event Processing in Action. Manning Publications Co.

Fontaine, M.D., Smith, B.L., 2005. Probe-based Traffic Monitoring Systems with Wireless Location Technology. Transportation Research Record 1925, 3–11.

Fontaine, M.D., Smith, B.L., 2007. Investigation of the Performance of Wireless Location Technology-Based Traffic Monitoring Systems. Journal of Transportation Engineering 133, 157–165.

Gama, J., Medas, P., Castillo, G., Rodrigues, P., 2004. Learning with Drift Detection, in: Proc. 17th Brazilian Symposium on Artificial Intelligence (SBIA), Springer. p. 286.

Gama, J., Rodrigues, P., 2009. An Overview on Mining Data Streams. Foundations of Computational Intelligence 6, 29–45.

Gama, J., Sebastião, R., Rodrigues, P.P., 2009. Issues in Evaluation of Stream Learning Algorithms, in: Elder IV, J.F., Fogelman-Soulié, F., Flach, P.A., Zaki, M.J. (Eds.), Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM. pp. 329–338.

Garcia-Molina, H., Ullman, J.D., Widom, J., 2009. Extended Operators of Relational Algebra, in: Database Systems: The Complete Book. Prentice Hall Press, Upper Saddle River, NJ, USA, 2 edition. pp. 213–222.

Geisler, S., Chen, Y., Quix, C., Gehlen, G., 2010a. Accuracy Assessment for Traffic Information Derived from Floating Phone Data, in: Proc. of the 17th World Congress on Intelligent Transportation Systems and Services.

Geisler, S., Quix, C., Gehlen, G.G., Jodlauk, G., 2009. A Quality- and Priority-based Traffic Information Fusion Architecture, in: Proc. of the 16th World Congress on Intelligent Transport Systems and Services.

Geisler, S., Quix, C., Schiffer, S., 2010b. A Data Stream-based Evaluation Framework for Traffic Information Systems, in: Ali, M., Hoel, E., Shahabi, C. (Eds.), Proc. of the 1st ACM SIGSPATIAL International Workshop on GeoStreaming, pp. 11–18.

Golab, L., Özsu, M.T., 2010. Data Stream Management. Synthesis Lectures on Data Management, Morgan & Claypool Publishers.

Han, J., Kamber, M., 2006. Classification and Predicition, in: Data mining: concepts and techniques. Morgan Kaufmann, pp. 285–382.

Hashemi, S., Yang, Y., Mirzamomen, Z., Kangavari, M., 2009. Adapted One-versus-All Decision Trees for Data Stream Classification. IEEE Transactions On Knowledge and Data Engineering 21, 624–637.

Horovitz, O., Krishnaswamy, S., Gaber, M.M., 2007. A Fuzzy Approach for Interpretation of Ubiquitous Data Stream Clustering and its Application to Road Safety. Intelligent Data Analysis 11, 89–108.

Hoyer, R., 2003. Hinweise zur Datenvervollständigung und Datenaufbereitung in verkehrstechnischen Anwendungen. Technical Report. Forschungsgesellschaft für Strassen- und Verkehrswesen. (in German).

Huber, W., 2001. Vehicle-generated Data for Aquiring Traffic Information. Ph.D. thesis. TU München.

Hulten, G., Spencer, L., Domingos, P., 2001. Mining Time-changing Data Streams, in: Proc. of the 7th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining, ACM. pp. 97–106.

Jain, N., Mishra, S., Srinivasan, A., Gehrke, J., Widom, J., Balakrishnan, H., Çetintemel, U., Cherniack, M., Tibbetts, R., Zdonik, S., 2008. Towards a Streaming SQL Standard. Proceedings of the VLDB Endowment 1, 1379–1390.

Jensen, C.S., Tradisauskas, N., 2009. Map Matching, in: Liu and Özsu [2009]. chapter M.

Kargupta, H., Sarkar, K., Gilligan, M., 2010. MineFleet®: An Overview of a Widely Adopted Distributed Vehicle Performance Data Mining System, in: Proc. of the 16th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, ACM. pp. 37–46.

Kerner, B.S., 2004. The Physics of Traffic. Springer.

Khan, A., 2007. Intelligent Infrastructure-based Queue-end Warning System for Avoiding Rear Impacts. IET Intelligent Transport Systems 1, 138–143.

Laskov, P., Gehl, C., Krüger, S., Müller, K., 2006. Incremental Support Vector Learning: Analysis, Implementation and Applications. The Journal of Machine Learning Research 7, 1909–1936.

Liu, L., Özsu, M.T. (Eds.), 2009. Encyclopedia of Database Systems. Springer.

Liu, Y., Choudhary, A., Zhou, J., Khokhar, A., 2006. A Scalable Distributed Stream Mining System for Highway Traffic Data, in: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (Eds.), Knowledge Discovery in Databases, Springer. pp. 309–321.

Nagel, K., Schreckenberg, M., 1992. A Cellular Automaton Model for Freeway Traffic. Journal de Physique 2, 2221–2229.

Oza, N., 2006. Online Bagging and Boosting, in: Proc. of the IEEE International Conference on Systems, Man and Cybernetics, IEEE. pp. 2340–2345.

Oza, N.C., Russell, S., 2001. Online Bagging and Boosting, in: Jaakkola, T., Richardson, T. (Eds.), Proc. of the 8th International Workshop on Artificial Intelligence and Statistics, Morgan Kaufmann, Key West, Florida. USA. pp. 105–112.

Patroumpas, K., Sellis, T.K., 2006. Window Specification over Data Streams, in: Current Trends in Database Technology - EDBT 2006 Workshops, pp. 445–464.

Pelossof, R., Jones, M., Vovsha, I., Rudin, C., 2010. Online Coordinate Boosting, in: Proc. of the 12th International Conference on Computer Vision, Workshops, IEEE. pp. 1354–1361.

Srivastava, U., Widom, J., 2004. Flexible Time Management in Data Stream Systems, in: Deutsch, A. (Ed.), Proc. 23rd ACM Symposium on Principles of Database Systems (PODS), ACM, Paris, France. pp. 263–274.

Stathopoulos, A., Karlaftis, M., 2003. A Multivariate State Space Approach for Urban Traffic Flow Modeling and Prediction. Transportation Research Part C 11, 121–135.

Stonebraker, M., Çetintemel, U., Zdonik, S.B., 2005. The 8 Requirements of Real-time Stream Processing. SIGMOD Record 34, 42–47.

Thakkar, H., Mozafari, B., Zaniolo, C., 2008. Designing an Inductive Data Stream Management System: The Stream Mill Experience, in: Proc. of the 2nd international workshop on Scalable stream processing system, ACM. pp. 79–88.

Tsymbal, A., 2004. The Problem of Concept Drift: Definitions and Related Work. Technical Report TCD-CS-2004-15. Informe técnico: Departament of Computer Science, Trinity College. Dublin, IE. `https://www.cs.tcd.ie/publications/techreports/reports`.

Wang, Y., Papageorgiou, M., 2005. Real-time Freeway Traffic State Estimation Based on Extended Kalman Filter: A General Approach. Transportation Research Part B: Methodological 39, 141–167.

White, C.E., Bernstein, D., Kornhauser, A.L., 2000. Some Map Matching Algorithms for Personal Navigation Assistants. Transportation Research Part C: Emerging Technologies 8, 91–108.

Wojnarski, M., Góra, P., Szczuka, M.S., Nguyen, H.S., Swietlicka, J., Zeinalipour-Yazti, D., 2010. IEEE ICDM 2010 Contest: TomTom Traffic Prediction for Intelligent GPS Navigation, in: Fan, W., Hsu, W., Webb, G.I., Liu, B., Zhang, C., Gunopulos, D., Wu, X. (Eds.), 10th IEEE Intl. Conf. on Data Mining – Workshop Proceedings (ICDMW), IEEE Computer Society, Sydney, Australia. pp. 1372–1376.