Learning Classifier Systems for Road Traffic Congestion Detection

Matthias Sommer and Jörg Hähner
Organic Computing Group, University of Augsburg, Eichleitner Str. 30, 86159 Augsburg, Germany

Keywords: Road Traffic, Congestion Detection, Learning Classifier Systems, Machine Learning, Reinforcement Learning.

Abstract: The increase in mobility leads to a higher number of kilometres driven per vehicle and more delay due to congestion which poses a recent and future problem. Congestion generates growing environmental pollution and more car accidents. We apply machine learning concepts to the task of congestion detection in road traffic. We focus on the extended classifier system XCSR, an evolutionary rule-based on-line learning classifier system. Experiments with real-world detector data demonstrate high accuracy of XCSR for congestion detection on interstates.

1 INTRODUCTION

According to several reports (Schrank et al., 2012; Lenz et al., 2010), the number of kilometres driven and the delay due to congestion increased over the last decades and this trend is assumed to last. Raising traffic volumes promote growing air pollution, a greater number of car accidents, and more traffic congestion. Intelligent incident management systems try to mitigate the negative effects of congestion. This includes the collection of sensor data, the detection or prediction of congestion, and the execution of actions for the congestion management. The detection process is often performed by automatic incident detection (AID) algorithms, e.g. by processing video image material from traffic surveillance cameras or by pattern recognition on sensor data. These classic approaches work well under certain conditions, but their performance is often strongly dependent on predefined thresholds and they are not able to adapt to new and previously unknown patterns at runtime. In this work, we apply machine learning concepts to the task of congestion detection on interstates. Learning classifier systems (LCS), such as the extended classifier system XCSR, resemble evolutionary rule-based machine learning techniques that have shown to work well for classification tasks (Bull, 2004). XCSR evolves new rules at runtime with the help of a genetic algorithm (GA), while also improving its accuracy over time via reinforcement of the existing rule set. Experiments with real-world detector data were carried out to investigate the performance of XCSR under real-world conditions. Support vector machines (SVM) have proven to be accurate classifiers for traffic congestion detection (Diamantopoulos et al., 2014; Šingliar and Hauskrecht, 2006). Consequently, we compare our approach to several representatives.

The remainder of this work is structured as follows. First, we provide a brief overview of the related work in this field. We move on, mapping the formal concept of congestion detection to machine learning problems. We introduce the reader to the fundamentals of learning classifier systems in general, and the XCSR in particular. Based on the theoretical concept, we present how LCSs can be practically used to tackle the congestion detection problem. Another machine learning concept, in particular SVMs, are consulted as a reference solution for the later evaluation. We conclude this work with a summary of our findings and an outlook on future work.

2 RELATED WORK

Incident detection is one of many components of advanced traffic management systems. (Ozbay and Kachroo, 1999) define it as “the process of identifying the spatial and temporal coordinates of an incident”. It is executed by automatic algorithms or by manual evaluation. Reliable detection mechanisms and fast clearance are important for mitigating the negative effects of incidents and congestion. Figure 1 depicts the typical flow of the incident management process (Deniz et al., 2012).

First, data from surveillance systems (e.g. CCTV

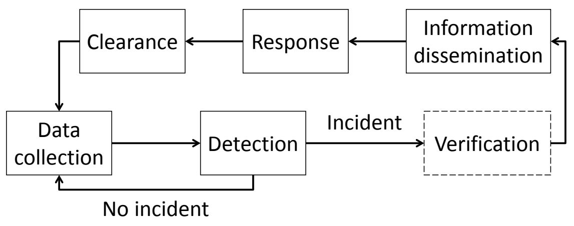


Figure 1: Typical flow chart of an AID system. The verification step is optional.

cameras or loop detectors) provides a situation description of the current traffic condition. Second, this data is usually sent to a central control centre where it is processed. The data analysis is often executed by automated incident detection algorithms. An incident is described by its type (recurrent, blockage, etc.), its exact location, its severity, and the time of occurrence. Third, the incident alarm can be verified by an operator, e.g. via surveillance cameras. Fourth, the edited information has to be disseminated among the traffic participants. The congestion management is usually done by traffic experts. Its strategies range from adaptation of signal plans, to re-routing of traffic by means of route recommendations via variable message signs, and radio broadcasts. Finally, clearance procedures are initiated to restore the undisturbed conditions as before the incident. In this work, we focus on the second step, presenting how learning classifier systems can be used to analyse traffic data to determine the presence of congestion.

Comprehensive reviews over congestion detection algorithms and detector technology are given by (Parkany and Xie, 2005) and (Mahmassani et al., 1999). The family of point-based algorithms is usually deployed on freeways (Yang et al., 2004). It can be separated into comparative algorithms, statistical processing, traffic modeling and theoretical algorithms, and advanced machine learning algorithms. Spatial measurement-based algorithms make use of CCTV cameras and image processing algorithms and are also used in urban traffic networks (Zhang and Xue, 2010). Congestion patterns are detected based on temporal and spatial differences of traffic parameters monitored by traffic sensors. The performance of many of these algorithms is strongly dependent on thresholds set during design time by traffic experts based on historic data. Furthermore, they are prone to obsolescence in case of changing traffic demands and are not able to learn new behaviour. In contrast, learning classifier systems evolve their knowledge at runtime, being able to adapt to a changing environment by learning new rules. Additionally, they can be trained upfront based on labelled training data.

3 CONGESTION DETECTION AS MACHINE LEARNING TASK

In its simplest form, congestion detection depicts a binary classification problem. In this case the two classes represent the presence or absence of congestion. Typically, the classes are imbalanced, meaning that the class representing free-flowing traffic has much more instances than the congested class. This imbalance and the resulting lack of instances makes the learning process more difficult. This two-class problem can be expressed more formally as

$$f(\vec{x}) \rightarrow c_i \in \{i = 1,2\} \qquad (1)$$

A feature vector $\vec{x} = \{x_1, x_2, \ldots, x_n\}$ is processed by a function f which maps the input variables to a specific class c_i. In case of congestion detection, $\vec{x}$ contains a defined set of traffic parameters describing the current traffic conditions. The goal is to fit a model that relates the observations in $\vec{x}$ to the correct class label c_i. Machine learning techniques use artificial intelligence to deduce a process, model, or function from observations to describe a certain behaviour (Alpaydın, 2008). A number of authors applied different machine learning algorithms to the problem of congestion detection, such as SVMs (Diamantopoulos et al., 2014; Šingliar and Hauskrecht, 2006), artificial neural networks (Srinivasan et al., 2004), and fuzzy logic algorithms (Brumback, 2009). Instead of just relying on one single technique, some researchers combine several methods, e.g. (Liu et al., 2014) use multiple naïve bayes classifiers. Some of these algorithms are able to learn new patterns and to improve their model at runtime (reinforcement learning), e.g. learning classifier systems. These algorithms choose and execute actions in reaction to the observations and adjust their parameters, and their internal and external model according to the feedback received.

4 LEARNING CLASSIFIER SYSTEMS IN A NUTSHELL

The *learning classifier system* (LCS) (Bull and Kovacs, 2005) is founding on the Holland's initial framework for classifier systems (Holland, 1986), resembling an evolutionary on-line machine learning technique that is designed for both single-step and multi-step problems. LCS combines ideas from evolutionary computing, reinforcement learning, supervised or unsupervised learning, and heuristics. This adaptive, rule-based system builds a descriptive model for the underlying observations. The knowledge base consists of a population of rules (or classifiers) which map situation descriptions to actions. An evolutionary algorithm evolves these rules in order to explore the problem space. The existing rules are rated upon their influence on the system or task. Therefore, rules that have shown to achieve good results have a higher possibility to be chosen again in later executions.

In 2000, Wilson proposed the *eXtended Classifier System for Real-valued inputs (XCSR)* (Wilson, 2000). Its advantage over the traditional LCS is its ability to take real-valued inputs which are usually found in real-world environments. XCSR tries to evolve accurate and maximally general classifiers that cover the state-action space of the underlying problem. A single XCSR classifier cl_j comprises a couple of attributes: 1) The condition C_j that determines a certain subspace of the problem space by encoding a geometric structure, 2) an action a that defines a reaction that can be executed on the environment, 3) the payoff prediction p_j estimating the payoff of the system in case C_j matches the current situation and its action was chosen, 4) an error estimate ε that reflects the mean absolute prediction error, and 5) a fitness value ϕ that can be roughly interpreted as an inverse of ε, which represents the accuracy of the prediction.

The learning process is as follows. First, the classifier system receives an input $\vec{x}$, representing the current environmental state. Second, based on this situation description, classifiers matching $\vec{x}$ are selected. Because different classifiers can represent different actions, an action selection process has to be executed. Within our scenario, an action resembles the class prediction for the current traffic condition. In case a classifier cl was chosen and the execution of its according action resulted in a positive influence of the environment, cl gains a positive reward. Otherwise, the rating of this classifier is reduced. The reinforcement process leads to better system performance over time. Additionally, the problem space is explored by creating new classifiers for previously unknown situations at runtime.

5 XCSR FOR CONGESTION DETECTION

Classifier systems have been successfully applied in a variety of real-world applications (Bull, 2004). To the best of the authors' knowledge, the distributed, adaptive control of signalisation is their only application within the traffic domain (Bull et al., 2004; Prothmann et al., 2008). In the following, we explain in detail how we adapt XCSR to the classification of traffic conditions. In the context of AID, the situation description is a vector $\vec{x} = (x_1, \ldots, x_n)$ of continuous, real-valued traffic parameters monitored by sensors, e.g. loop detectors or CCTV cameras. Therefore, $\vec{x}$ represents the current traffic condition at an intersection or section. The action a is the estimated congestion classification (0 for congested and 1 for free-flowing). The underlying task is mapped to a single-step problem, as the according reward (the actual state of traffic) for a is returned in the next time step.

5.1 The Main Loop

The classification process for the current traffic condition works as follows. In every time step t, values monitored by a traffic detector are retrieved. We simulate the sensor input by reading the next line from a data set. This input is then converted into a feature vector $\vec{f_t}$. In case we do not want to use every available sensor value from $\vec{f_t}$, we define which features to include and which ones to omit (we may only be interested in the average speed and occupancy). XCSR demands the input values to be in the range of $[0;1[$, thus all components of $\vec{f_t}$ have to be normalised to this value range. This is no limitation to its application since the upper limits for the traffic parameters, such as occupancy or speed, of a given road can be estimated. The resulting vector $\vec{s_t}$ is then given to XCSR for classification.

First, the condition of each classifier cl of the population $[P]$ is compared to the current input $\vec{s_t}$. In case cl matches the external input, it is added to the *match set* $[M]$. If $[M]$ consists of too less classifiers (compared to a predefined threshold), the GA is triggered to create new, random classifiers matching $\vec{s_t}$. This *covering* process is executed to cover previously unknown situations and to enable XCSR to offer a prediction for the current situation. The newly created classifiers are initialised according to pre-defined values for the prediction p, the prediction error ε, and the initial fitness F. Afterwards, a fitness-weighted average P_{a_k} of the predictions p_j for each action a_k represented in $[M]$ is computed as

$$P_{a_k} = (\sum_j \phi_j p_j)/(\sum_j \phi_j P_{a_k}) \qquad (2)$$

P_{a_k}, the system prediction of action a, is added to the *prediction array* $[P]$. Here, $[P]$ consist of two entries, one for each possible classification. Then, an action a_i is chosen from $[P]$ based on the *action selection* regime. This selection can either be random (*Exploration*), probabilistic, or deterministic (*Exploitation*). Finally, the action set $[A]$ consists of the subset of classifiers of $[M]$ having the chosen action. After executing a, the prediction, prediction error, and fitness of each classifier in $[A]$ are updated based on the received reward (*Reinforcement*). In our scenario, the reward is either 1000 for a correct classification or 0 for a false classification.

5.2 Training and Rule Discovery

XCSR is an on-line learning algorithm. For a valid comparison with other algorithms, we simulate an off-line training phase before testing. During training, XCSR *explores* the situation space. Afterwards, the gained knowledge is *exploited*. The training is supervised, thus, each situation vector of the training set has to be classified into one of two classes, congested or free flowing. The training examples are given in the form $\{(x_i, y_i)\}$ such that x_i is the feature vector and y_i its class label. Amongst other factors, the exploration rate strongly depends on the execution rate of the GA and the chosen action selection regime.
In certain intervals, defined by a parameter θ_{GA}, XCSR tries to explore the search space by creating new rules. This *discovery* process is executed by the GA. Two classifiers are chosen probabilistically from the latest action set based on their fitness. An offspring is created by crossing the two parents based on a two-point crossover of their conditions, and then mutating the condition and action with a certain probability. A classifier is mutated by adding to or subtracting a random offset from its condition representation. The prediction is set to the mean of the parents' prediction values. The fitness and the prediction error are also set to the mean of the parents values, multiplied with reduction factors α, respectively p_{red}. As a result, the two new offspring classifiers are added to the population.
Usually, the action selection method during the exploration phase is random. For faster convergence, we utilise a fitness-proportionate action selection, also known as roulette wheel selection. After the execution of the chosen action, its reward is returned, and a reinforcement of the selected classifiers takes place.

5.3 Testing

After completing the off-line training phase, XCSR relies on its previously learned knowledge. For the on-line application, we switch the action selection regime to a deterministic best-action selection. The best action is represented by the classifier with the highest fitness-weighted score in $[P]$. The GA is no longer executed in certain intervals. However, covering is still executed in case of unknown situations or missing actions in $[M]$. We always want at least one classifier in $[M]$ representing one of the two traffic classifications. Thereby, new classifiers can still be created and added to the population.

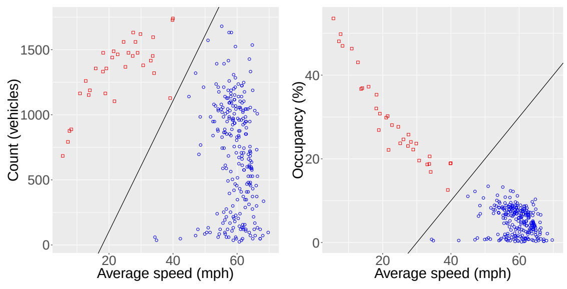
5.4 Parameter Study

The commonly used settings for the learning parameters of XCS is given by (Butz and Wilson, 2002). Starting with these initial parameter settings, a small parameter study for the most important learning parameters was conducted: $\beta = \{0.1, 0.2, 0, 5\}$, $\theta_{GA} = \{1, 5, 15, 25, 50\}$, $s_0 = \{0.1, 0.2, 0.5\}$, $m_{cs} = \{0.1, 0.2\}$, $m_{ob} = \{0.1, 0.2, 0.4\}$, $r_{ob} = \{0.1, 0.2, 0.4\}$, $p_X = \{0.2, 0.3, 0.5\}$, and $p_M = \{0.04, 0.05, 0.06\}$. The best performance was achieved with the following settings (see Table 1): $\beta = 0.2$, $\theta_{GA} = 5$, $p_X = 0.3$, $p_M = 0.05$, and the unordered bound representation with $m_{ob} = 0.2$ and $r_{ob} = 0.2$. An increased learning rate *beta* allows the system to adjust classifiers faster but makes it more sensitive to temporary peaks. A decrease in the number of executions of the GA leads to raising variance of the results.

Table 1: Initial parameter settings for the most important learning parameters of XCSR.

N	Max. number of micro-classifiers	800
β	Learning rate for p, ε, and ϕ	0.2
ϕ_{init}	Initial classifier fitness	0.01
ε_{init}	Initial classifier prediction error	0.0
p_{init}	Initial classifier prediction value	10.0
δ	Classifier fitness deletion threshold	0.1
ε_0	Classifier accuracy threshold	10
θ_{sub}	Classifier subsumption threshold	20
θ_{del}	Class. experience deletion threshold	20
θ_{GA}	GA application interval	5
p_X	Crossover probability	0.3
p_M	Mutation probability	0.05
α	Fitness reduction factor	0.1
p_{red}	Prediction reduction factor	0.25
s_0	Centre spread factor	0.2
m_{cs}	Mutation prob. for centre spread	0.1
m_{ob}	Mutation prob. for (un)ordered bound	0.2
r_{ob}	Covering prob. for (un)ordered bound	0.5

A visualisation of the state space helps to estimate the complexity of the underlying problem. Figure 2 depicts the dependency between volume and speed (Figure 2(a)), and speed and occupancy (Figure 2(b)) for a random day. The black line exemplary visualises a possible linear separation between states that are categorised as congested or not congested. The separation between congested and free-flowing traffic conditions is rather clear for most situations. Congested conditions can be assumed in case the average speed falls below a certain threshold while the occupancy or the number of vehicles increases.



(a) Volume and speed. (b) Speed and occupancy.

Figure 2: Scatter plots showing the state space for different traffic variables measured at highway I35E, detector 2447 (red squares depict congested, blue circles free flowing conditions).

Further performance can be gained by adjusting the number of traffic parameters within the feature vector. On the one hand, more features can describe the underlying dynamics of traffic more precisely. On the other hand, a higher number of features expands the search space drastically. This leads to longer exploration durations while needing more classifiers to describe the respective feature space.

6 EVALUATION

6.1 Experimental Setup: Traffic Data

The evaluation was done with ten real-world data sets provided by the Minnesota Department of Transportation (MDoT)[1]. The data was recorded by inductive loop detectors in the vicinity of Minneapolis (Figure 3), averaged over five minute intervals, resulting in 2016 data points per week. Each data point contains the time of recording, average speed, volume, occupancy, and density. The congestion labels were annotated by hand whereas a sudden speed drop and raise in occupancy indicated the presence of congestion.

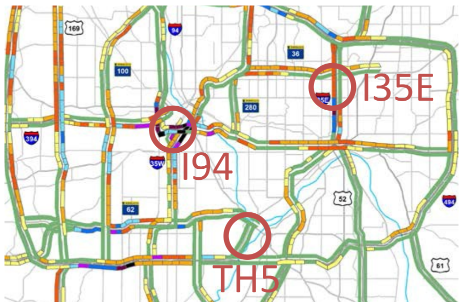


Figure 3: Locations of the selected detector stations in Minneapolis, U.S.

[1]http://dot.state.mn.us/tmc/trafficinfo/developers.html

The monitoring locations and dates are taken randomly: Interstate I35E, June/July 2013 (detectors 2442, 2443, 2444, 2447, 2448); Interstate TH5, December 2015/January 2016 (detector 1577); and Interstate I94, May/June 2015 (detectors 569, 365, 366, 367). Traffic on I35E and I94 shows some typical seasonal behaviour. Congestion usually occurs during rush hours in the morning and evening on work days. In contrast, traffic on TH5 exhibits stop-and-go behaviour in the early hours during work days. MDoT defines congestion as traffic flowing at speeds below 45 miles per hour. Our data sets exhibit congested conditions between 2% and 29% of the time. This class imbalance (between congested and free-flowing data points) is commonly found in data from real-world environments. Each data set was split into three weeks for training (6048 data points) and one week for testing (2016 data points).

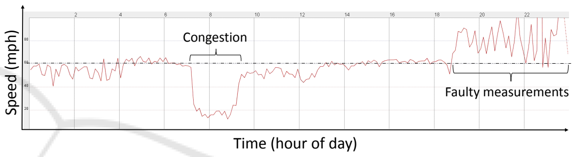


Figure 4: Traffic flow profile for arterial I35E, detector station 2447, 2013-06-12.

Figure 4 shows a representative traffic flow profile on I35E, measured by detector 2447 on Wednesday, 2013-06-12. Usually, the monitored speed fluctuates around the recommended limit of 60 mph. The plot depicts a typical weekday morning rush hour from 7.30 a.m. to 9.30 a.m. where the average speed is reduced to 20 mph. The detector station recorded faulty measurements from 7 p.m. to midnight, a typical problem of real-world data from inductive loop detectors (Parkany and Xie, 2005). We did not remove faulty measurements to evaluate how XCSR and SVM deal with this problem.

6.2 Support Vector Machines for Congestion Detection

Support vector machines (SVM) (Ben-Hur and Weston, 2010) have proven to provide good generalisation and convergence for classification tasks. Their goal is to approximate the optimal hyperplane, accurately separating the state space into two distinct classes. Their performance is dependent on a coefficient C defining the margin between classes and the kernel hyper-parameter γ of the gaussian kernel handling the non-linear classification. Large values for γ and C give a low bias and high variance because the cost of misclassification gets stronger penalized. In contrast, small values result in higher bias and lower variance.

The following SVM implementations from the *JKernelMachines* framework (Picard et al., 2013) were chosen as references for the evaluation: LaSVM, LaSVM-I, and SDCA. *LaSVM* (Bordes et al., 2005) is an efficient SVM solver that uses on-line approximation. It is able to handle noisy data sets using less memory than other state-of-the-art SVM solvers. *LaSVM-I* (Ertekin et al., 2011) is an optimisation of LaSVM. It filters outliers based on approximating non-convex behaviour in convex optimisation. LaSVM-I proves to be faster in terms of training times, needing fewer support vectors, and offering only slightly worse accuracy. Stochastic dual coordinate ascent (*SDCA*) (Shalev-Shwartz and Zhang, 2013) is a method for solving large-scale supervised learning problems formulated as minimisation of convex loss functions. It executes iterative, random coordinate updates.

We conducted a small parameter study for C, γ, and the type of kernel. The gaussian chi-squared kernel wa chosen as it showed to be faster than the gaussian kernel with L2 distance, providing similar accuracy. It is calculated as

$$k(x,y) = 1 - \sum_{i=1}^{n} \frac{(x_i - y_i)^2}{\frac{1}{2}(x_i + y_i)} \tag{3}$$

Test runs with $C = \{1, 10, 100, 500, 1000\}$ and $\gamma = \{0.01, 0.1, 0.5, 1\}$ indicated that $C = 10$ and $\gamma = 0.01$ yield good results without overfitting the model.

6.3 Experimental Results

All following experimental results are average over ten runs, executed on the previously introduced data sets. According to (Parkany and Xie, 2005), speed and occupancy are chosen about 80% of the time in terms of congestion detection in traffic management centres in the U.S. Following this advice, we present the experimental results for the feature vector comprising these two parameters.

The choice between several approaches is often dependent on multiple factors. One aspect is their performance on historical data which we evaluate and discuss in the following section. Other factors include the runtime, the interpretability by humans, and the convenience to configure the respective technique.

Runtime. Table 2 shows the mean runtime and standard deviation averaged over ten execution runs. The evaluation was done on an Intel i7 dual-core with 2.6 GHz and 8 GB RAM. The SVM variants are used as is from the JKernelMachines framework (Picard et al., 2013). Considering execution times, XCSR has a clear benefit over SVM, having much lower runtime, both for training and testing (each data set has 4032 data points). To speed up the training process of the SVM variants, the number of training epochs E can be reduced. The following speed-up can be achieved by reducing E from five (default value) to one: LaSVM (8.4 sec.), LaSVM-I (2.6 sec.), SDCA (2.0 sec.). However, these results have to be interpreted with caution. Each data point of the data set was given one-by-one to XCSR (on-line learning), whereas the SVMs were given the training set as a whole, speeding up the learning process drastically as the model is computed only once. In fact, if the SVMs are trained using on-line learning, adjusting the internal model after every time step, the execution times are significantly longer, e.g. LaSVM-I needs 12.5 minutes and SDCA 5.5 minutes for a single training run (4032 data points, $E = 5$).

Table 2: Average runtime (and standard deviation) in seconds for the training phase and the test phase.

Method	Training	Testing
LaSVM	14.2 (10.3)	0.8 (0.3)
LaSVM-I	9.5 (14.3)	1.7 (1.5)
SDCA	5.6 (0.5)	7.6 (0.2)
XCSR	2.6 (0.4)	0.5 (0.2)

Configuration. Mostly, XCSR offers fairly good performance out-off-the-box using its standard parameter settings. Still, a fair amount of parameter studying is needed to find the optimal settings. In this aspect, SVM is quite simple to configure since it only requires hyper-parameters C and γ and the number of epochs E to be set, as well as the kernel to be chosen.

Interpretability. Another aspect is the understandability of the model. SVMs resemble a very flexible method. Still, their interpretability is very low as the support vectors are difficult to analyse or to visualise (James et al., 2013). XCS is designed to be interpretable by humans, while still being flexible. Classifiers can be added and adapted during runtime, and their respective values, action, and condition are easily understandable.

Number of Classes. We formulate congestion detection as a binary classification problem. Considering XCSR, increasing the number of classes (e.g. tentative congestion or faulty detector data) is no problem as only the number of distinct classifier actions has to be adjusted. The complexity of XCSR's implementation stays the same. In contrast, SVMs are usually two-class classifiers. For multi-class tasks, decomposition methods such as one-against-all or one-against-one are used. Solving a multi-class SVM in one step results in a much larger optimisation problem (Hsu and Lin, 2002).

6.3.1 Measuring Classifier Accuracy

Given a labelled data set, the following four basic measures are usually used for statistical analysis of classifiers:

- True positives (TP): Number of correct results where roads are predicted to be congested.
- True negatives (TN): Number of correct results where roads are classified as free flowing and there is actually no congestion.
- False positives (FP): Number of falsely predicted congested roads, whereas traffic flows freely.
- False negatives (FN): Number of falsely predicted free flowing roads, whereas the road is actually congested.

In other words, TP and TN describe the accuracy of the classifier (the predicted class label matches the actual classification). FP and FN measure the error rate of the evaluated classifier. Naturally, high detection rates and a minimal number of false alarms is desired. However, these two performance measures are not independent. The number of false alarms can easily be reduced by decreasing the sensitivity of the detection algorithm. Still, this will result in poor detection rates. In contrast, increasing the detection rate DR ($DR = TP/(TP+FN)$) will also increase the false alarm rate FAR ($FAR = FP/(TN+FP)$).

As shown in Table 3, XCSR has a low FAR of 0.26% and a high DR of 95.5%. The average number of FP and FN is rather low. The FAR and DR of the SVM variants are as follows: LaSVM (FAR = 0.21%, DR = 90.0%), LaSVM-I (FAR = 0.43%, DR = 74.3%), and SDCA (FAR = 1.4%, DR = 91.5%).

The following metrics are described in terms of TP, TN, FN and FP. The accuracy A specifies the number of correct results as

$$A = \frac{TP+TN}{N} \tag{4}$$

where N is the total number of classified situations. A classifier who simply classifies all situations as free flowing achieves high accuracy since the probability that traffic is congested is generally much lower than free flowing traffic. The precision P is calculated as

$$P = \frac{TP}{TP+FP} \tag{5}$$

An algorithm who predicts few or no congestion may result in high precision since the number of FP is minimised. In general, high precision means that the classifier returns more correct than wrong predictions. The recall R measures the proportion of positives that are correctly identified by

$$R = \frac{TP}{TP+FN} \tag{6}$$

High recall can easily achieved by classifying all situations as congested. The F-measure considers both recall R and precision P. It is calculated as

$$F = \frac{2PR}{P+R} \tag{7}$$

Finally, the specificity SP measures the proportion of negatives that are correctly identified by

$$SP = \frac{TN}{FP+TN} \tag{8}$$

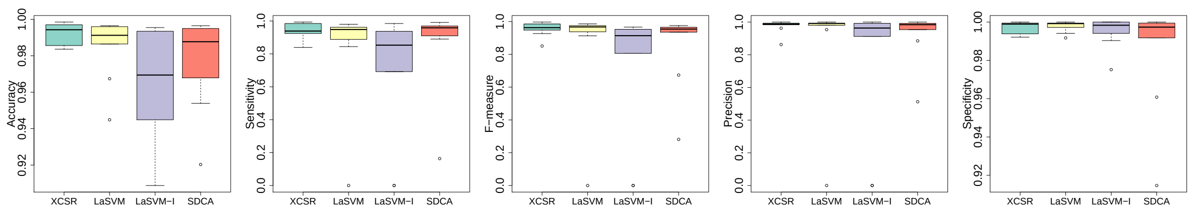
Figure 5 presents the results for these measures. The box plots show the statistical distribution of the average classification accuracy. The bottom and top of the box represent the first and third quartiles, and the band inside the box represents the median. Outliers are indicated by separate points. In general, all approaches had very high accuracy (an average of 97% and above), classifying most of the congested situations as congested, and most of the not congested situations as free flowing. Figure 5(d) indicates that LaSVM-I misclassified too many situations as congested. In general, LaSVM-I performed slightly worse compared to the other algorithms. Most of the outliers are caused by the TH5 data set which has relatively few congested situations. However, all machine learning techniques seem to struggle in learning to differentiate its feature space, due to the lack of instances belonging to the congested class. Although, XCSR has its lowest values for recall (0.84), precision (0.86), and the F-measure (0.85), it still offers a fair performance for the TH5 data set. LaSVM and LaSVM-I were not able to learn the task for this data set as they simply classify all situations falsely as not congested. On average, XCSR has better results for accuracy, recall, and F-measure than the SVMs, offering similar performance for precision and specificity.

6.3.2 Learning Behaviour of XCSR

In the following, we evaluate the learning behaviour of XCSR. We measure the system error, the population size, and the fraction of correct classifications in every execution. Figure 6 shows how XCSR is able to improve its performance over time. The vertical dotted line marks the end of the training phase after 6000

Table 3: Confusion matrix reporting the average number (and standard deviation σ) of true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN) of XCSR for the ten test data sets.

		Actual class		
		Congested	Free flowing	Total
Prediction	Congested	248 ($\sigma = 170$) (TP)	5 ($\sigma = 5$) (FP)	253
	Free flowing	12 ($\sigma = 8$) (FN)	1751 ($\sigma = 170$) (TN)	1763
	Total	260	1756	2016



(a) Accuracy. (b) Recall. (c) F-measure. (d) Precision. (e) Specificity.

Figure 5: The box plots show the statistical distribution of the average classification accuracy for XCSR, LASVM, LASVM-I, and SDCA (left to right) considering occupancy and speed.

time steps. The points of each curve are the fraction of the last 50 executions, averaged over ten runs. The fraction correct is the percentage of correct classifications in the last 50 executions. The system error is calculated as the absolute difference between the actual reward and the system prediction P_a of the selected action, divided by the maximal reward (1000). The population curve shows the average number of micro classifiers normalised to the range of [0;1]. The population size was roughly between 3400 and 5400. Accordingly, we chose 6000 as the maximum number of micro classifiers within the population. The curve shows how the number of classifiers continuously increases during the training phase. XCSR applied covering between 15 and 35 times (average: 24.4). The number of GA executions ranges from 325 to 777 (average: 494.5), which translates to one GA execution every 12th step during the training phase. Due to the explorative behaviour of XCSR during training, the fraction of correct classified instances and the system error fluctuate more during this phase.

7 CONCLUSION

We applied the extended classifier system XCSR to the task of detection of congestion patterns on interstates. The evaluation was done with real-world data monitored by inductive loop detectors located in Minneapolis. In conclusion, it can be noted that XCSR is able to evolve accurate classifiers, offering reliable accuracy for the classification of traffic conditions. Compared to three different types of support vector machines, XCSR offers competitive performance.

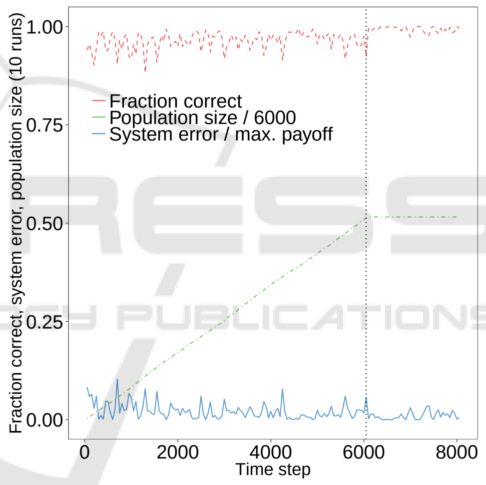


Figure 6: Average fraction correct, system error, and population size for XCSR (feature vector: occupancy and speed).

Furthermore, XCSR is significantly faster in terms of runtime for training and testing. In contrast to the representation of support vectors, XCSR's rule base of classifiers of is easily interpretable by humans. We plan to extend the binary classification problem by introducing additional classes, such as tentative congestion or continuing congestion. Instead of classifying the current situation, XCSR can be adapted to predict the upcoming traffic conditions. Furthermore, we want to investigate the performance of XCSR for urban congestion detection at intersections and sections, following ideas from (Klejnowski, 2008).

ACKNOWLEDGEMENT

The authors would like to thank Roman Sraj for his contribution in the scope of his bachelor's thesis.

REFERENCES

Alpaydın, E. (2008). *Maschinelles Lernen*. Oldenbourg.

Ben-Hur, A. and Weston, J. (2010). *Data Mining Techniques for the Life Sciences*, chapter A User's Guide to Support Vector Machines, pages 223–239. Humana Press.

Bordes, A., Ertekin, S., Weston, J., and Bottou, L. (2005). Fast kernel classifiers with online and active learning. *Journal of Machine Learning Research*, 6:1579–1619.

Brumback, T. E. (2009). *A Mathematical Model for Freeway Incident Detection and Characterization: A Fuzzy Approach*. PhD thesis, University of Alabama.

Bull, L., editor (2004). *Applications of Learning Classifier Systems*. Springer.

Bull, L. and Kovacs, A. (2005). *Foundations of Learning Classifier Systems*, volume 183, chapter Foundations of Learning Classifier Systems: An Introduction, pages 1–17. Springer.

Bull, L., Sha'Aban, J., Tomlinson, A., Addison, J. D., and Heydecker, B. (2004). Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In *Applications of Learning Classifier Systems*, pages 276–299. Springer.

Butz, M. and Wilson, S. (2002). An algorithmic description of XCS. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 6:144 – 153.

Deniz, O., Celikoglu, H. B., and Gurcanli, G. E. (2012). Overview to some incident detection algorithms: A comparative evaluation with istanbul freeway data. *Proc. of 12th Int. Conf. Reliability and Statistics in Transportation and Communication*, pages 274–284.

Diamantopoulos, T., Kehagias, D., Knig, F. G., and Tzovaras, D. (2014). Use of density-based cluster analysis and classification techniques for traffic congestion prediction and visualisation. In *Transp. Research Arena Proceedings*.

Ertekin, S., Bottou, L., and Giles, C. (2011). Nonconvex online support vector machines. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 33(2):368–381.

Holland, J. H. (1986). A mathematical framework for studying learning in classifier systems. *Phys. D*, 2(1-3):307–317.

Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *Trans. Neural Networks*, 13(2):415–425.

James, G., Witten, D., Hastie, T. J., and Tibshirani, R. J. (2013). *An Introduction to Statistical Learning*. Springer.

Klejnowski, L. (2008). Design and implementation of an algorithm for the detection of disturbances in traffic networks. Master's thesis, University of Hannover, Institute for Systems Engineering.

Lenz, B., Nobis, C., Köhler, K., Mehlin, M., Follmer, R., Gruschwitz, D., Jesske, B., and Quandt, S. (2010). Mobilität in Deutschland 2008. *DLR-Forschungsbericht*.

Liu, Q., Lu, J., Chen, S., and Zhao, K. (2014). Multiple naïve Bayes classifiers ensemble for traffic incident detection. *Mathematical Problems in Engineering*, 2014.

Mahmassani, H. S., Haas, C., Zhou, S., and Peterman, J. (1999). Evaluation of incident detection methodologies. Technical report, Center for Transportation Research, University of Texas.

Ozbay, K. and Kachroo, P. (1999). *Incident Management in Intelligent Transportation Systems*. Artech House ITS library. Artech House.

Parkany, E. and Xie, P. C. (2005). A complete review of incident detection algorithms & their deployment: What works and what doesn't. Technical report, Fall River, MA: New England Transp. Consortium.

Picard, D., Thome, N., and Cord, M. (2013). Jkernelmachines: a simple framework for kernel machine. *Journal of Machine Learning Research*, 14(1):1417–1421.

Prothmann, H., Rochner, F., Tomforde, S., Branke, J., Müller-Schloer, C., and Schmeck, H. (2008). Organic control of traffic lights. In *Proc. of the 5th Int. Conf. on Autonomic and Trusted Computing*, volume 5060 of *LNCS*, pages 219–233. Springer.

Schrank, D., Eisele, B., and Lomax, T. (2012). *2012 Urban mobility report*. Texas A&M Transp. Institute. http://mobility.tamu.edu/ums/.

Shalev-Shwartz, S. and Zhang, T. (2013). Stochastic dual coordinate ascent methods for regularized loss. *Journal of Machine Learning Research*, 14(1):567–599.

Šingliar, T. and Hauskrecht, M. (2006). Towards a learning incident detection system. In *Proc. of the Workshop on Machine Learning for Surveillance and Event Detection at the 23rd Int. Conf. on Machine Learning*.

Srinivasan, D., Jin, X., and Cheu, R. L. (2004). Evaluation of adaptive neural network models for freeway incident detection. In *IEEE Transaction on Intelligent Transp. Systems*, volume 5, pages 1–11.

Wilson, S. (2000). Get real! xcs with continuous-valued inputs. In *Learning Classifier Systems*, volume 1813 of *LNCS*, pages 209–219. Springer.

Yang, X., Sun, Z., and Sun, Y. (2004). A freeway traffic incident detection algorithm based on neural networks. In Yin, F.-L., Wang, J., and Guo, C., editors, *Advances in Neural Networks - ISNN 2004*, volume 3174 of *LNCS*, pages 912–919. Springer.

Zhang, K. and Xue, G. (2010). A real-time urban traffic detection algorithm based on spatio-temporal OD matrix in vehicular sensor network. *Wireless Sensor Network*, 2(9):668–674.