Prediction of Congested Traffic on the Critical Density Point using Machine Learning and Decentralised Collaborating Sensors

Wouter Labeeuw[1], Kurt Driessens[2], Danny Weyns[2], Tom Holvoet[2], Geert Deconinck[1]

[1] Departement of Electrical Engineering,
{wouter.labeeuw,geert.deconinck}@esat.kuleuven.be
[2] Department of Computer Science,
{kurt.driessens,danny.weyns,tom.holvoet}@cs.kuleuven.be
Katholieke Universiteit Leuven, Belgium

Abstract. In this paper we discuss short term traffic congestion prediction, more specifically, prediction of the sudden speed drop when traffic resides at the critical density point. We approach this problem using standard machine learning techniques combining information from multiple sensors measuring density and average velocity. The model used for prediction is learned offline. Our goal is to implement (and possibly update) the predictive model in a multi-agent system, where coupled with each sensor, there is an agent that monitors the condition of traffic, starts to collect data from other sensors located nearby when necessary and is able to predict local sudden speed drops so that drivers can be warned ahead of time. We evaluate Gaussian processes, support vector machines and decision trees not only limited to predictive accuracy, but also the suitability of the learned model in the setup as described above, i.e., keeping in mind that we want the warning system to be decentralized and want to ensure scalability and robustness.

Keywords: road traffic, short term velocity drop prediction, critical density, machine learning, multi-agent system

1 Introduction

The bulk of existing work on traffic prediction focuses on density [7, 1] and occupancy [16] prediction. The systems that use these predictions are employed to traffic management purposes [5].

In this paper, we focus on short-term predictions of traffic velocity instead. Our goal is to design a distributed system that can predict the sudden, local drop of speed that marks the start of congested traffic. If local predictions can be made with sufficient accuracy, we can warn oncoming traffic of the expected trouble ahead of time. Specifically, we are interested in the question: "Can we predict local future velocity drops at the critical density point in a decentralized way?"

The critical density point is the density at which the behavior of traffic is least predictable. Traffic flow can be distinguished into regimes [15]: free-flow, capacity-flow, congested, stop and go and jammed traffic. In free-flow traffic, vehicles can travel freely at their desired speed. When more and more vehicles join in, free-flow traffic gets denser until it reaches the capacity-flow, i.e., the maximum number of vehicles able to maintain free-flow traffic. The border between capacity-flow and congested traffic is the critical density point and is situated around 25 vehicles per kilometer per lane [20]. In this paper, a practical attempt to predict the future velocity at this transition point is made using standard machine learning algorithms. The experimental setup consists of sensors and agents that try to predict the future average velocity within their own range, using self collected data and data from the surrounding sensors. The experiments are build upon traffic simulation software [17] in which an intelligent driver model realistically directs the behavior of the individual vehicles [18] and use the machine learning suite Weka [21] for the machine learning components.

2 Predicting traffic congestion

Figure 1 represents the average velocity in meters per second as measured by a sensor over time and illustrates the prediction task we focus on in this work. The plot of the average velocity will be slightly different for sensors at different locations, but the sudden drops look similar in each plot. We consider the sudden drop to be the most useful information to predict, since it can be used to notify drivers of an upcoming dangerous situation. If drivers can be notified 1 km in advance that a strong reduction in speed is expected, it could significantly lower the probability of an accident happening due to distractions or a loss in concentration. Note that we focus on the region where the velocity stays more or less constant until the drop. The area indicated by B represents stop and go waves. Once traffic is in a stop and go wave, it will stay in a stop and go wave if the density does not drop drastically. The backpropagation of such a wave is easier to predict and has been handled in previous work [18]. In this work, we will focus on predicting the transition from A to B when, from the view of the driver, the congestion is hardest to anticipate.

The Dutch traffic information service divides traffic congestion into three subclasses: slow traffic (min 25 km/h and max 50 km/h for minimum 2km), stationary traffic (max 20 km/h) and a combination of both [9]. The chosen cut off point for the velocity is 7 m/s, which more or less equals 25 km/h. The vertical line in figure 1 indicates the time of this drop below 7 m/s.

We will approach this problem using supervised machine learning. Supervised machine learning builds predictive models using labelled training examples. This model can then be used to make predictions about the labels of previously unseen examples. The task of our learning problem is: “Given data from a local sensor and its surrounding sensors, learn a model that correctly predicts the velocity drop”. For this feasibility study, we use the Weka tool [21], which implements a number of different machine learning algorithms.

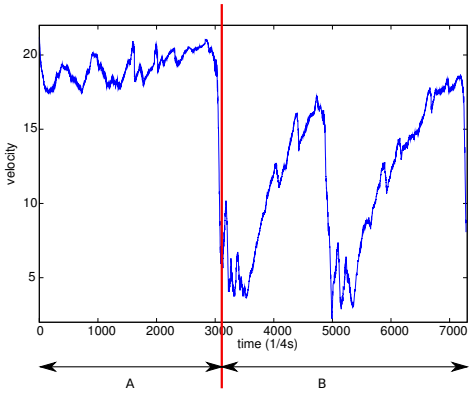

Fig. 1. Velocity drop

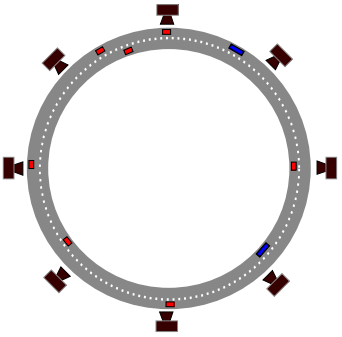

Fig. 2. Camera setup

2.1 Setup

The setup for the experimental study is reasonably simple. We used Treibers software [17] to simulate a ring road with an average speed and density sensor every kilometer. The range of each sensor was set to 350 m as is standard for traffic cameras [10]. A ring road was chosen, because it models a straight road with infinite history which is long enough to get congested traffic. Figure 2 illustrates the used setup. While this setup represents only an initial study, it allows us to illustrate a number of important issues. Learning data was collected by taking a snapshot of sensory information 4 times each second. To this snapshot we added the local average velocity as measured a fixed time t into the future. Different values of t were tried as will be discussed later.

2.2 Predicting numerical velocity

In a first step, we try to predict the future velocity using a regression algorithm, i.e., a learning algorithm that predicts a real value as an outcome. The learning algorithm we use was Gaussian processes with a radial basis function (RBF) kernel. Gaussian processes represent a strong baseline for regression [8]. Non-linear kernels, such as the RBF kernel, perform well when dealing with a large number of learning instances with a relatively low number of features [6, 3]. In short, Gaussian processes implement a non-parametric Bayesian technique. Bayesian regression techniques assume a prior distribution over the function hypothesis space (usually over the parameter vector defining the function) and calculate a posterior distribution using Bayes rule and the available learning data. Instead of assuming a prior over the parameter vectors, Gaussian processes assume a prior over the target function itself. We refer to [14] for a more elaborate discussion of the workings of Gaussian processes

As the covariance function required by Gaussian processes, we use a RBF kernel defined as $K(x_i, x_j) = exp(-\gamma||x_i - x_j||^2)$ (illustrated as $K(x_i, x_j) =$

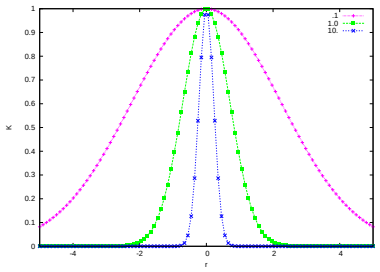

Fig. 3. Radial Basis Function

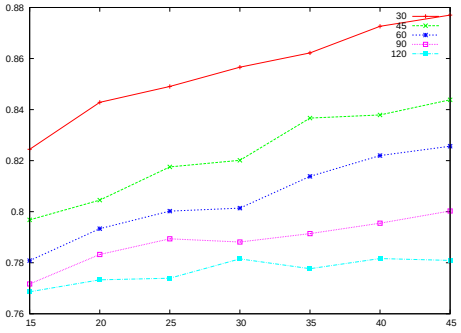

Fig. 4. Correlation: Change in time after drop

$exp(-\gamma r^2)$ in figure 3) [6]. The γ parameter controls the width of the kernel and thereby the amount of generalization used by the Gaussian processes. Higher γ values result in less generalisation. A small search over the γ of the RBF showed that γ equalling 1 gave better correlation results over other values. The correlation is an indication in how much two coeffients are related to each other. In this case how the predicted values and the actual values relate. All other learning experiments with Gaussian processes will use this value for the γ parameter.

Gaussian processes also allow specification of a measurement noise level. The way the sensors measure velocity as the average speed of passing traffic, gives rise to a natural measurement noise level, i.e., the standard deviation of the measured velocities.

Since we are trying to make predictions about the transition between free flow traffic and congested traffic, we need to collect data from both the A and the B region of Figure 1. Since we don't want to predict stop and go traffic, the amount of data from region B must be limited. Figure 4 shows how collecting more data after the transition influences the measured correlation. More data collected after the transition raises the prediction correlation. If data is collected for a time period longer then 30 seconds, the measured data originates from the B area which seems easier to predict. In this view, a collection time close to but below 30 seconds (the lowest studied prediction time) seems to be a good choice.

To test the need for a MAS approach in our sensor system, we first compared correlation results of using data from only the local sensor to data collected from a total of nine surrounding sensors (local, plus 4 sensors before and 4 sensors behind the target point). The training examples contain the velocity and the density from the participation sensors and the future velocity as the label to be predicted. Table 1 shows that (not surprisingly) using information from multiple sensors gives better correlation results than information from only the local sensor, which supports the need for multi-sensor cooperation in this type of prediction task. The gain of using multiple sensors ranges from small at

a prediction time of 30 seconds, i.e. when predicting what the measured velocity will be 30 seconds in the future, to substantial when the prediction time increases.

Table 1. Correlation results

Pred. time	1 camera	9 cameras
30	0.81954	0.84855
60	0.68813	0.78329
120	0.43739	0.73997

Table 2. History

Pred. time	$\Delta t = 0$	$\Delta t \neq 0$
30	0.84855	0.85594
45	0.79709	0.82672
60	0.78329	0.80898
90	0.75415	0.79424
120	0.73997	0.78293

As measured velocity is sequential data where trends in the measurements might be highly informative we also investigated the predictive gain of past measurements or "sensory history". Using data from Δt time ago together with current measurements gives rise to better correlation results than using current information alone. Tests show that the exact value of Δt is of less importance. The correlation results are similar for Δt ranging from 5 to 30 (averaged in Table 2) and significantly better than the results of using no history ($\Delta t = 0$ in the table). The higher the prediction time, the bigger the gain from using a history.

Figure 5 shows a detailed plot of real future velocities (x-axis) versus predicted velocities (y-axis) for a prediction time of 60 seconds and $\Delta t = 10$. Data was collected until 25 seconds after the velocity drop. Area C at the top right shows a high correlation between real and predicted at high velocities. These are the recorded velocities before the transition, when traffic is still in free-flow. After the velocity suddenly drops, the model has more difficulties to make correct predictions. In area A and B, representing stationary and slow traffic respectively, future velocities are often predicted too high.

While using Gaussian processes delivers reasonable predictive results they do present a different difficulty. The learned model is large and consumes a large amount of memory and when making predictions, the model has to perform a lot of computations. This makes the model less suitable for agent purposes. Other regression techniques are expected to, at best, deliver only comparable results.

2.3 Predicting classified velocity

Since regression turns out to be hard, we study the use of classification for our congestion prediction problem. Classification is concerned with the prediction of nominal labels instead of numeric ones. In our problem, we consider three different types of traffic, and thus three possible labels: stationary, congested and normal traffic. These types are based upon the Dutch traffic information service. Stationary means speed below 7 m/s, congested is below 14 m/s, normal traffic is above 14 m/s. These are also the areas marked in figure 5.

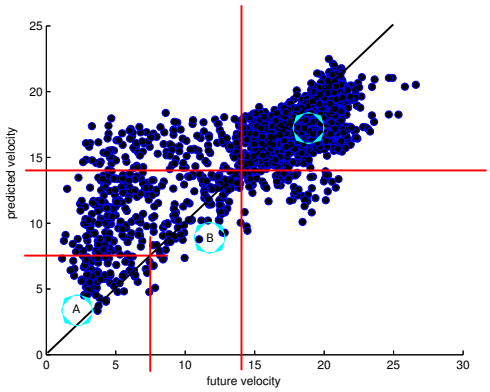

Fig. 5. Predicted velocity versus actual future velocity

The performance of the model will be examined by following quality evaluators: [21]

Accuracy is the overall probability that the model makes a correct prediction.

Recall is the percentage of actual positive instances that are predicted as positive, also called the *true positive rate*. In our traffic warning system, this will indicate the percentage of velocity drops that are predicted as such.

Precision is the percentage of positive predictions that are real positive instances. In our warning system, this gives an indication of the number of false alarms, i.e., precision = 1−(% of false alarms).

Since we are dealing with a 3-class prediction problem, we define a positive prediction to be a correctly classified instance, meaning e.g. '*normal*' has been classified as '*normal*' and a negative prediction as a wrongly classified prediction, meaning e.g. '*congested*' has been classified as either '*normal*' or '*slowdown*'.

Support Vector Machines In a first step we used a classification approach, very related to the Gaussian processes used for regression before: support vector machines (SVM). In short, SVM's classify examples using a linear decision boundary. In most cases, linear decision boundaries are not expressive enough to separate examples from different classes and therefor SVM's use a non-linear transformation on the input space to a new, high-dimensional, space. This way, a linear model in the new space can be a non-linear decision boundary in the original space [4]. The algorithm we used for training the SVM is Sequential Minimal Optimisation (SMO). SMO is very fast with linear kernels and reasonably fast with non-linear kernels. The memory footage grows linear with the training set, which means large training sets are possible [11]. For the input space transformation, we again used an RBF-kernel. To determine the best values for the parameters C (SMO) and γ (RBF) we again performed a grid search. $C = 10$

and $\gamma = 1$ gave the best results and all following experiments were preformed using these values.

Table 3 shows the influence of history on the evaluators. While recall is hardly influenced by adding history, it does make a difference for the accuracy and precision of the model. When no history is used ($\Delta t = 0$ in the Table), the results are somewhat worse than with the use of history. Since again the results for various values of $\Delta t \neq 0$ are similar, Table 3 shows averaged evaluation values.

Table 3. SVM: impact of the use of history

	$\Delta t = 0$			$\Delta t \neq 0$		
Pred. time	Accuracy	Recall	Precision	Accuracy	Recall	Precision
30	93.050	98.714	95.664	93.618	98.641	96.417
45	92.116	98.621	94.163	92.553	98.216	95.459
60	90.517	98.518	92.695	91.481	97.942	94.527
90	88.605	97.395	92.035	90.088	97.501	93.562
120	86.402	97.091	89.810	88.433	97.398	91.917

The model learned by SMO is small enough to be implemented into simple agents. The size ranges from 200 kB to 1 MB. One problem with using SVMs in an application as critical as traffic warning systems is that the learned model is pretty much a black box, and that it is next to impossible to interpret its decision strategies. More importantly however, the true positive rate on congested and slowdown are quite low. For example with a prediction time equal to 60 and history $\Delta t = 10$, the true positive rate for the prediction of normal traffic is 99.6 %, while the true positive rate for congested traffic is only 55.6 % and for slowdown only 17.6 %. While SMO is quite good at predicting normal traffic, it has significantly more difficulties with congested traffic and performs even worse on slowdown. Since these are exactly the conditions our warning system is looking for, this will not do.

Decision Trees To alleviate both problems indicated above, we tried decision trees on the same classification problem. Decision trees are a machine learning technique that employs a "Divide and Conquer" approach [12]. Decision Trees can easily be converted into rules and thus, the decisions they make can be interpreted and checked by a human, which is an advantage in critical applications such as traffic control.

We used the Weka variant of the C4.5 algorithm [13] as a decision tree learner. C4.5 has been a benchmark algorithm for a long time. In Weka, the algorithm is implemented as J48 [21]. Test results show that sensory history doesn't influence the quality evaluators significantly. Table 4 shows the results for different prediction times. Not surprisingly, the predictive performance drops with increase prediction time.

Table 4. DT: results

Pred. time	Accuracy	Recall	Precision	Rules
30	91.648	97.625	95.368	68.235
45	90.129	96.346	94.532	87.520
60	88.928	95.755	93.796	100.24
90	86.316	94.435	92.283	124.52
120	83.970	91.195	89.112	144.82

The results are close to those obtained using the support vector machine. Using only a reasonably small number of rules (indicated in Table 4) the model performs comparable to the more computation intensive SVM model. However, with respect to the precision for the important classes, i.e. *congested* and *slowdown*, decision trees actually perform better. Using a prediction time equal to 60 and $\Delta_t = 10$ as as before, the true positive rate for '*normal*' is slightly worse than the results of SMO: 95.8% but those for '*congested*' traffic and '*slowdown*' actually improve significantly. Congested traffic has a true positive rate of 75.1% and slowdown of 53.4%. This means that fewer incorrect warnings will be given using decision trees than using the SVM model. An extract of the tree build for this test is shown below:

```
PrevVelocity4 <= 13.141108
|   PrevVelocity5 <= 17.9016
|   |   PrevVelocity3 <= 9.702755
|   |   |   PrevVelocity1 <= 15.582357
|   |   |   |   PrevVelocity1 <= 12.889484: slowdown
|   |   |   |   PrevVelocity1 > 12.889484: normal
|   |   |   PrevVelocity1 > 15.582357: congested
|   |   PrevVelocity3 > 9.702755
|   |   |   PrevVelocity4 <= 12.566312: congested
|   |   |   PrevVelocity4 > 12.566312
|   |   |   |   PrevDensity0 <= 85.714286
|   |   |   |   |   PrevDensity3 <= 45.714286: congested
|   |   |   |   |   PrevDensity3 > 45.714286: slowdown
...
```

3 Feasibility study

We want to make scalability and robustness important design criteria for our congestion warning system and therefor approach it as a distributed multi-agents system. In a first, simple design, each agent of the multi-agent system consists of a sensor, a repository and a decision maker. The sensor collects the necessary traffic information for prediction purposes and stores it into the agent's repository. The decision maker periodically checks the repository. When a certain density is encountered, the decision maker also collects information from other neighboring agents. The decision maker predicts the traffic speed state within the near future using the machine learned model, the data in the repository and the data received from the other agents. This decentralized approach ensures scalability. Adding more agents with the same machine learned model is easier than adjusting a central decision controller. There is no single point of failure. One failing agent will at most influence its eight neighboring agents.

3.1 Practical approach

We have evaluated the multi-agent system approach in a simulation setting [17]. In this simulation, the sensors which represent the sensory part of the agents are modeled as virtual camera's. The camera's are placed equidistant at 1 km from each other and their range is 350 m.

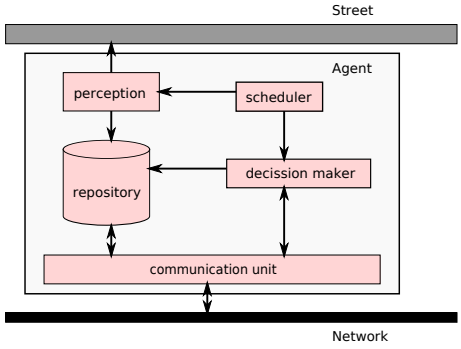

Fig. 6. Camera structure

The agent architecture can be found on Figure 6. The *perception* unit (the virtual camera) collects information from the street and calculates the density and intensity. This information is stored into the *repository*. The *scheduler* periodically stimulates the camera to take a snapshot, perform calculations and store the results. The *decision maker* checks the stored data in the repository. If the density is within the critical density, the decision maker will request the data from the neighboring agents. The collected data is then presented at the machine learned model, in order to predict the future velocity. The collection of data from the other agents, happens through the *communication unit*.

The repository stores and keeps history of the density and intensity measured by the perception unit. It also stores the neighboring agents. This neighbor information is useful for the decision maker. It ensures the data from the correct neighbors is collected. The neighboring agents are found using a bootstrap process. When the camera starts up, the camera broadcasts its existence and asks for other cameras within its range. The scheduler periodically stimulates the repository to check the correctness of the position of the relevant neighbors.

The number of agents is limited by the bandwidth of the communication network. If all agents would request specific neighbor information at the same time, it could overload the network. Alternatively, all information could be broadcasted and logged at each agent, but this would undo the local view of the agents.

3.2 Placing the model (and learner) inside the agent

The learned model from section 2 is part of the decision maker. The type of information requests to the neighboring agents will depend on the model, e.g.,

some models need historical data while others don't. For prediction purposes, Weka itself can be included. Switching models is easy in this approach, since it only requires the loading of a different model. The disadvantage is that Weka is somewhat big to be included inside an agent. Decision trees, for example, can easily be converted to rules and implemented inside the decision maker. Weka straightforwardly supports the generation of rules (java code) from the learned decision trees. The advantage of including Weka as part of the agent is that the prediction model could be adapted online as discussed in future work.

4 Related work

Related work on traffic and congestion prediction is quite extensive and a large scale overview is well beyond the extend of this paper. A lot of existing work focuses on long term predictions, such as the work by Yasdi [22], that predicts the density on weekly, daily and hourly basis through the use of a neural network. The goal of the predictions is use them to reroute traffic to keep the density on the roads below the critical point and avoid congestion formation completely. The decisions are made at a central point. In contrast, we focus on short term predictions of the average velocity on a short road segment in a distributed system.

Abdulhai et al. [1] try to predict the density within minimum 30 seconds an maximum 15 minutes, also using a neural network. The used densities all lie within free flow boundaries. The results show that the farther into the future one tries to predict, the more the neural network tends to predict the average density. The data is collected from 9 loop detector stations and the prediction is done centralized.

A very different approach for the prediction of congested traffic is an ant based system [2]. Every vehicle leaves a trail of pheromones, based on the traffic information. In this setup, vehicles themselves are able to predict congestion from the information of preceding cars. The density and velocity aren't measured directly but predicted through the accumulation of virtual pheromones.

Huisken et al. [7] compare time series analysis (ARMA) and neural networks (MLF) performance in predicting congestion through density estimation on a relatively short-term time scale (5 - 15 minutes) and conclude that neural networks gave the best results.

Taylor et al [16] try to predict the volume and the occupancy of traffic using a multilayer perception network. The prediction happens one minute in advance on data from 6am until 9am during weekday's. The performance of prediction is tested using the mean squared error. The neural network seems to perform quite well.

In contrast to all related work discussed above, we predict actual short term velocity drops when traffic is at the critical density point, instead of estimating density related values.

5 Conclusions

In this paper, we evaluated the use of machine learning techniques for the prediction of the average velocity of traffic at the critical density point. The machine learning models were trained using data from multiple cameras. The models were evaluated on their correctness and interpretability. Decision Trees gave the best results in both.

The warning system itself was designed as a multi-agent system. The agents contain the learned model, check their environment using a camera view for critical densities and when traffic resides at the critical density point, start to collect data from other cameras and use the learned model for short term velocity or congestion predictions. The model will trigger an alarm when congestion eminent and warn upcoming drivers that traffic is slowing down. These alarms and models can hopefully be used to lessen the occurrence of stop and go traffic.

In the current setup, failure of an agent influences the predictive capacities of the neighboring agents. To increase robustness, probability nodes instead of decision nodes can be used inside the decision tree when decision critical data is unavailable. Instead of deciding which path to take based on the (unavailable) data, the probability of the missing condition can be used to combine the decisions made by the child branches. This would mean that failure of one agent does not imply that eight other agents are also unable to make any predictions.

The proposed structure should be tested on real traffic data instead of simulation data. Considering that the simulator implements realistic driving behaviors, we expect at least some of our conclusion to carry to the real world.

Another interesting next step is letting the agents change their environment by warning drivers or implementing some speed limitations. Changing the environment will automatically cause the learned models to become wrong, raising the need for online learning. By integrating the machine learning algorithm within the agents, the model can be optimized online to account for local road specifics. Utgoff [19] presented an algorithm for incremental decision tree learning. Since agents collect data at the critical density point for prediction no extra communication overhead is caused to collect online training data.

References

1. Baher Abdulhai, Himanshu Porwal, and Will Recker. Short-term freeway traffic flow prediction using genetically optimized time-delay-based neural networks. In *Transportation Research Board*, Washington D.C, 1999.
2. Yasushi Ando, Yoshiaki Fukazawa, Osamu Masutani, Hirotoshi Iwasaki, and Shinichi Honiden. Performance of pheromone model for predicting traffic congestion. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 73–80, New York, NY, USA, 2006. ACM Press.
3. C. Campbell. An introduction to kernel methods. In Robert J. Howlett, Lakhmi C. Jain, and Janusz Kacprzyk, editors, *Radial basis function networks 1: recent developments in theory and applications*, pages 155–192. Physica Verlag Rudolf Liebing KG, Vienna, Austria, 2001.

4. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.
5. Josefa Z. Hernández, Sascha Ossowski, and Ana García-Serrano. Multiagent architectures for intelligent traffic management systems. *Transportation Research Part C: Emerging Technologies*, 10(5-6):473 – 506, 2002.
6. C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science and Information Engineering, National Taiwan University, Taipei, 2003.
7. G. Huisken and A. Coffa. Short-term congestion prediction: comparing time series with neural networks. In *Road Transport Information and Control, 2000. Tenth International Conference on (Conf. Publ. No. 472)*, pages 66–69, 2000.
8. D. J. C. MacKay. Introduction to Gaussian processes. In C. M. Bishop, editor, *Neural Networks and Machine Learning*, NATO ASI Series, pages 133–166. Kluwer Academic Press, 1998.
9. VerkeersInformatieDienst Nederland. http://www.verkeersinformatiedienst.nl/fileproblematiek.html.
10. Traficon NV. Traficon camera specification: http://www.traficon.be/pagenode.jsp?id=7&type=ProductCategory.
11. John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in kernel methods: support vector learning*, pages 185–208, Cambridge, MA, USA, 1999. MIT Press.
12. J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
13. J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.
14. Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.
15. Maerivoet S. and De Moor B. Traffic flow theory. Technical report, ESAT-SCD (SISTA) K.U.Leuven, Leuven, Belgium, July 2005.
16. C. Taylor and D. Meldrum. Freeway traffic data prediction using neural networks. In *Vehicle Navigation and Information Systems Conference, 1995. Proceedings. In conjunction with the Pacific Rim TransTech Conference. 6th International VNIS. 'A Ride into the Future'*, pages 225–230, Jul-2Aug 1995.
17. Martin Treiber. http://www.vwi.tu-dresden.de/ treiber/MicroApplet/.
18. Martin Treiber, Ansgar Hennecke, and Dirk Helbing. Congested traffic states in empirical observations and microscopic simulations. *Physical Review E*, 62:1805, 2000.
19. Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.
20. Verkeerscentrum Vlaanderen, Departement Leefmilieu en Infrastructuur, AdministratieWegen en Verkeer, Vuurkruisenplein 20, 2020 Antwerpen *MINDAT — Databank ruwe verkeersdata Vlaams snelwegennet — 2001 en 2003*, February 2003.
21. Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2 edition, 2005.
22. R. Yasdi. Prediction of road traffic using a neural network approach. *Neural Computing & Applications*, 8(2):135–142, May 1999.