Predicting Traffic Volume using User-generated Data

Vorhersage von Verkehrsaufkommen durch benutzergenerierte Daten
Bachelor-Thesis von Timo Nolle
01.08.2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
@Group

Predicting Traffic Volume using User-generated Data
Vorhersage von Verkehrsaufkommen durch benutzergenerierte Daten

Vorgelegte Bachelor-Thesis von Timo Nolle

Prüfer: Prof. Dr. Johannes Fürnkranz
Betreuer: Dr. Frederik Janssen, Dr. Immanuel Schweizer

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich, die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 01.08.2013

(Timo Nolle)

Contents

Zusammenfassung

Starker Verkehr auf den Straßen ist heutzutage ein gängiges Problem. Viele Städte nutzen Erkennungssysteme um Straßen mit hoher Auslastung erkennen zu könnnen, aber diese Systeme haben meist hohe Anschaffungskosten. In dieser Arbeit wollen wir einen Ansatz präsentieren, welcher die aktuelle Verkehrslage auf Basis von Schalldruckmessungen, welche von Smarpthones gemessen wurden, einschätzt. Wir nutzen maschinelle Lernalgorithmen um die aktuelle Verkehrssituation in eine von vier verschiedenen Stufen (frei, fließend, stockend und Stau) einzustufen. Wir nutzen die Daten von Induktionsschleifen der Stadt Darmstadt zur Erstellung einer Trainingsmenge für den Algorithmus. Induktionsschleifen werden zur Zählung von Kraftfahrzeugen eingesetzt, dabei erkennen die Schleifen das Metall, was sich über ihnen befindet. Die Daten der Induktionsschleifen mussten in eine Verkehrsstufe übersetzt werden, um die Trainingsmenge zu erstellen. Wir präsentieren in dieser Arbeit drei verschiedene Ansätze zur Berechnung einer Verkehrsstufe basierend auf Daten von Induktionsschleifen. Diese Ansätze wurden auf Basis von ungefähr 25,000 Schalldruckmessungen an den Straßen Darmstadt getestet. Dabei waren die erstellten Modelle in der Lage 80% der Situationen korrekt in eine der vier Verkehrsstufen einzuordnen. Dieser Ansatz zeigt, dass die Nutzung der Lautstärke an einer Straße als Grundlage für die Einschätzung von Verkehr praktikabel ist.

Abstract

Road congestion is a common problem, which cities try to fight by using detection systems. These systems are very expensive and not every city can afford these. In this work we present a system that uses sound pressure data measured by smartphone microphones to predict the current traffic situation on the road that those noise was captured on. This system uses machine learning to classify the traffic situation by learning on training data that was created by the inductive loops in Darmstadt. Inductive loops count the number of cars on a road by detecting metal that passes over them. In order to make this data usable for the machine learning we had to assign the inductive loop data to one of four traffic levels (free, flowing, congested, and traffic jam), which is then be used as the classification class. In this work we developed three different approaches to calculate a traffic level from inductive loop data. Using these approaches, and based on approximately 25.000 sound pressure measurements, we were able to correctly classify the traffic level in 80% of the test cases. This implies that using the sound pressure as a basis for a traffic level classification can be viable.

1 Introduction

Road congestion is getting more and more problematic nowadays. As the number of cars in the world grows every year, the traffic on the streets gets harder to control. People want to avoid heavy traffic whenever possible and warning systems for traffic congestion are inevitable. Real time traffic information is of great value when trying to avoid highly congested roads. Measurement systems that produce real time traffic information are often lacking in mobility and have high installation costs. The approach given in this thesis has nearly no costs and can easily be moved from one place to another.

The state of the art method for measuring traffic volume are inductive loops. These loops can count the amount of cars which drive on a road by recognizing the metal that passes over them. Unfortunately, these loops have high installation costs and have to be built into the road. Not every road can be equipped with such inductive loops, which is why a low budget system is desirable. The installation of just one inductive loop can cost several thousand euros.

Within this work we make use of mobile sensors, specifically smartphones. Smartphones get more and more common these days and their computing capacity is increasing rapidly. Nearly every smartphone comes with a GPS (Global Positioning System) antenna, which can be used to track the position of that sensor. Furthermore, every smartphone is naturally equipped with a high quality microphone, that can be used to measure the sound pressure level. When calibrated beforehand the accuracy of these microphones is sufficient for our purposes. The sound pressure depends on the amount of cars and the noise of their engines. We can use that correlation between traffic and sound pressure to predict the actual traffic condition on a road. By using a mobile sensor network consisting of smartphones we are able to cover a wide area, whereas inductive loops cover only the intersections of roads.

Throughout this work we will present an approach for assigning a traffic level (e.g., empty, free flowing, congested, traffic jam) to the road noise that traffic situations cause. As the basis for this assignment we use sound pressure measurements that captured this traffic noise. This approach uses machine learning to classify the traffic level and the inductive loops data was used as the training data for the machine learner. In order to create suitable training sets we had to calculate a traffic level from the inductive loop data, which can be used as the classification class for the machine learning algorithm.

This work used the results from Dimitar Goshev's "Road Traffic Monitoring with Location-aware Sound Sensors" [1] and refined the traffic level calculation. Furthermore, we adapted the machine learning to our new method of calculation and used a bigger training set then [1]. This thesis will give three different approaches for the determination of a traffic level from inductive loop data.

The aim of this work is to show that such a machine learning model can create satisfying results. This means that the algorithm is capable of predicting one of four different traffic levels, which correctly represents the current traffic situation by just using the sound pressure volume from that road. Three different machine learning algorithms were used to train a model for the classification of the traffic level. Together with the three approaches of calculating the levels, this gives a total of nine different models that were evaluated within this work. We were able to reach an accuracy of over 80%, i.e., the learned model was able to correctly classify 80% of the test cases, which were generated by cross validation.

We will start by giving a short introduction to machine learning. Then we will explain the used system and its architecture. The main part of this work is Section 5 in which we will explain our approaches to the traffic level calculation from inductive loop data. Then we will present three different approaches for the traffic level calculation and some observations we made while working on the problem. In the end we will evaluate these approaches and give the conclusions.

2 Related Work

The idea of using sound pressure measurements for traffic monitoring has been researched quite frequently in the last years. A low cost alternative to the state of the art inductive loop is very favorable for cities with low budgets. Many cities struggle with the increasing amount of cars on their streets, but they do not have the budget to install high end traffic monitoring systems.

In [2], Rijurekha Sen, Bhaskaran Raman and Prashima Sharma present a technique that uses differential Doppler shift to determine the speed of vehicles. They used the honking sound of vehicles, which are very frequently used in India. Two recording devices that are placed a few meters apart from each other are used to calculate the speed of a vehicle by using the delay from the honk sound arriving at both locations. Using this estimated vehicle speed distribution they were able to correctly classify the traffic situation with an accuracy of 70%. This means deciding whether or not the traffic is flowing or congested. The solution given by Rijurekha Sen, Bhaskaran Raman and Prashima Sharma is only able to assign a traffic situation to one of two classes (flowing and congested).

Rijurekha Sen, Pankaj Siriah, Bhaskaran Raman expanded the previously mentioned approach in [3]. By using the same recording devices, but using a mobile data connection to transmit the data to a server, they were able to calculate the traffic situation in real time. They also used machine learning algorithms for the classification of the traffic situation. In comparison to their previous approach they were able to raise the accuracy of the model from 70% to 90%. This solution however, can also only decide between two traffic states, congested and free-flow.

Prashanth Mohan, Venkata N. Padmanabhan and Ramachandran Ramjee also used smartphones to monitor road traffic conditions in [4]. In their paper from 2008 they present "Nericell", a system that does not only use the microphone but also the accelerometer of a smartphone to calculate the traffic quality on a road. They used the accelerometer to detect the road quality, by detecting potholes in the ground. A rapid change of the position can indicate such potholes. Furthermore, they used the accelerometer for the detection of breaking events. Breaking events are a good indicator for bad traffic quality or road congestion. They also used the microphone of the smartphone for the detection of honk sounds. Using all this information they were then able to detect low quality roads.

Dimitar Goshev has developed a system that uses the data from "da_sense" to classify traffic situations in [1]. He used data from inductive loops to label a training set and used sound pressure measurements as the basis for the classification. The classification class is a traffic level that represents the current traffic situation. These traffic levels are calculated using a simple clustering approach. He also states that the training set he used was very small, which is why he could not give a certain conclusion.

In this work we take the results from [1] and refine the traffic level calculation as well as the machine learning to fit our expectation of traffic levels. Furthermore, we evaluate the approach on a bigger training set of approximately 25.000 measurements.

3 Machine Learning

This work makes use of machine learning algorithms that learn to predict the traffic level when given some sound pressure data. Machine learning is very common in many modern applications. This section will give a quick introduction to the world of machine learning.

Arthur Samuel said in 1959 that machine learning is the “field of study that gives computers the ability to learn without being explicitly programmed”. Peter Flach defines machine learning as “the systematic study of algorithms and systems that improve their knowledge or performance with experience” in his book “Machine Learning: The Art and Science of Algorithms that Make Sense of Data” from 2012 [5]. Both definitions are equally appropriate, despite the first one being much older. This section is mainly based on the book from Peter Flach.

3.1 What is Machine Learning

The best way to introduce someone to the world of machine learning is by giving an example of a common application. Many people use it every day without noticing. Most spam filters are based on machine learning. The spam filter uses emails that were identified as spam by the user as training examples to learn what differentiates spam from non-spam email. Referring to the definition from [5] the experience would be the learning from some correctly labeled training data and the performance would be the ability of the filter to identify spam emails.

The basic task of a machine learning algorithm is to generalize a task, by observing a number of examples, so that it can be adopted to any new given case. A machine learning algorithm should be capable of correctly classifying any new previously unseen task. In order to do so it is inevitable that it generalizes from the given training examples, otherwise the algorithm could overfit to the problem, which means the learned model only works on the training examples, but not for new unseen cases. A model can also underfit the problem, which means that it does not even correctly classify the given training data. We have to find the soft spot between over- and underfitting in order to get a accurate model.

3.1.1 Supervised Machine Learning

There are two main kinds of machine learning systems, which will be presented here. In fact, there are more, but we focus on the two most commonly known. One is called supervised machine learning and the other unsupervised machine learning. Supervised means that the machine learning algorithms require some guidance in form of some examples that are already classified correctly. Unsupervised algorithms on the other hand do not need already classified data. It is sufficient to just provide example data without any previous labeling.

Machine learning algorithms learn from so called training sets. These training sets basically consist of a list of feature vectors, which in turn hold the different features. There are two different ways of learning. One is batch learning, where the algorithm is provided with one training set of any size. The other one is incremental learning, in which case the algorithm learns from a constant flow of new training examples. In our case we use batch learning. A feature is some interesting information about the task that shall be automated. In the spam filter example this could be something like a boolean that is true whenever an email contains the word “Viagra” for example. A feature can also be the amount of words that are contained in one email. Lets say we use two features for the spam filter example. One being the previously mentioned “Viagra” boolean and another one that does the same for the word “Lottery”. For an email that contains both those words the feature vector would look like this $F = (1, 1)^T$, where 1 stands for true and 0 for false. The task of the machine learning algorithm now is to decide whether such a feature vector stands for spam or non-spam. In supervised machine learning this requires the human to supply a training set, that consists of feature vectors that also include the expected classification, that is whether or not the given feature vector represents a spam email or not. We do so by adding a feature to the end of the feature vector, which represents the expected classification class. In our case the feature vector would change to $F = (1, 1, 1)^T$, because we expect the machine learning algorithm to learn that the feature vector $(1, 1)^T$ stands for spam. One feature vector is also often called a training instance or a training example.

After the learning process the machine learning algorithm returns a model that can classify new given feature vectors, without the need for the expected value. This model generalizes the given task and is now applicable to new instances. The main difference between different machine learning algorithms is the way the model is built. We will explain three different machine learning algorithms in the next section.

In unsupervised machine learning the algorithm is not supplied with a training set where the feature vector contains the expected classification value. As all the algorithms used in this thesis use supervised learning strategies, the unsupervised approach is not explained here. Information on unsupervised machine learning can be found in [5].

3.2 Used Algorithms

Within this thesis we use three different supervised machine learning algorithms: a decision tree algorithm , a rule based based algorithm, and a support vector machine. The support vector machine was chosen because it was used in [1]. Support vector machines, however, are mathematically very complex, which is why we chose to use two easier concepts for the comparison with this heavy weight approach. These three algorithms will be presented in the next three sections.

3.2.1 Decision Trees

Decision tree algorithms work by building a tree from the sample data which is then used to classify new cases. For our spam filter example such a tree could look as shown in Figure 3.1. We also used the two features from before in that figure. Such decision trees are parsed from top to bottom. The model starts at the root rule and then moves along the tree according to the results the rules give on a new instance. When the model arrives at a leaf it classifies the leaf's class.

The decision tree from Figure 3.1 would first check the email for the word "Viagra" and then for the word "lottery". When "Viagra" was already found "lottery" will not be checked and the classification will directly be "spam". Only when the email does not contain the word "Viagra", the model checks for the presence of the word "lottery" and if this is found it also classifies the instance as "spam". When none of the words are found the model will classify the email as "non-spam".

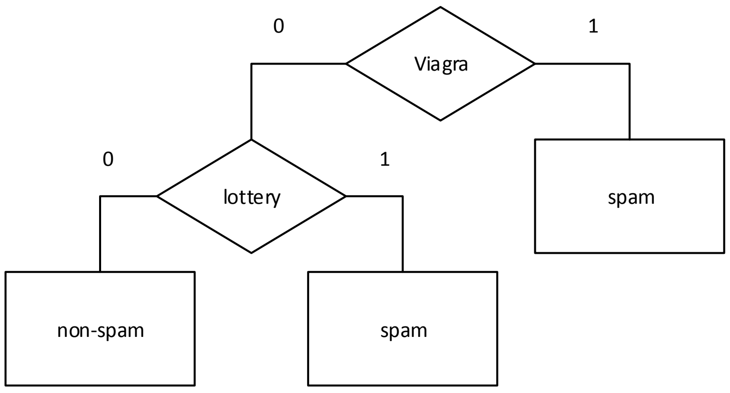


Figure 3.1: An example of a decision tree.

Now we explain how such trees are learned by the algorithm. Usually these trees are built from top down. Decision rules are generated from the given feature vectors in the training set. The feature "contains viagra" for example is a boolean feature, which means that possible outcomes are "true" and "false". This feature can be used as the root for the decision tree and the edges will be the possible outcomes of the feature. When a feature is numeric this is usually done with intervals. The algorithm generates these rules for every feature in the training set and then decides how to build them into a decision tree.

In order to decide which rules are better than others the algorithm needs a heuristic that compares two rules and gives a measure on how good they are. The best rule is a rule that perfectly divides the classes, so that only class 1 examples fall into the one subtree and only class 2 examples into the other (when the task is binary). Such rules have to be ranked higher than rules that do not contribute this much. A good heuristic that does this is called entropy.

Entropy basically measures how much a new rule orders the training set, or more specifically how much information is contained inside the subtree with that rule as the root. When a subtree only contains examples of one class, the rule is obsolete, hence the entropy value will be 0. Let S be the training set, $p_\oplus$ the proportion of class $\oplus$ examples, and $p_\ominus$ the proportion of class $\ominus$. The formula for the entropy is as in Equation 1.

$$E(S) = -p_\oplus \cdot \log_2 p_\oplus - p_\ominus \cdot \log_2 p_\ominus \tag{1}$$

If we take our example rule "contains Viagra" again and lets say the subtree contains 3 $\ominus$ (spam) examples and 2 $\oplus$ (non-spam) examples. The resulting entropy value is

$$E = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\cdot\log_2\left(\frac{3}{5}\right) = 0.971.$$

The entropy only gives this value for one specific rule outcome, but not for an entire feature (with all outcomes). To compute the entropy for an entire feature we have to average all possible outcomes and weigh the single subtrees by their size. Equation 2 shows this, where F is the chosen feature.

$$I(S,F) = \sum_i \frac{|S_i|}{|S|}\cdot E(S_i) \tag{2}$$

Using both the entropy and the average weighted entropy we can compute a value called Information Gain for a feature. This is defined as shown in Equation 3.

$$\text{Gain}(S,F) = E(S) - I(S,F) \tag{3}$$

To add a new rule to the decision tree, the algorithm computes the Information Gain and compares it to the other possible rules. The rule with the highest gain is chosen for the decision tree. This is done until every leaf only contains examples of one class.

3.2.2 Rule Based Learning

In Rule Based Learning the model at the end consists of a list of rules. This is basically the same as a decision tree, but the single rules are not connected to each other by edges. In fact every decision tree can be transformed into an equally expressive rule set. In our spam filter example one rule could look like this (contains Viagra) $\rightarrow \ominus$.

A new rule should increase the number of covered examples without decreasing the already covered ones. In other words, it should increase the number of covered examples of one class without decreasing the other class. We use a value called Laplace heuristic, which can be seen in Equation 4. This heuristic is close to one with $\oplus \rightarrow \infty$ and $\ominus \rightarrow 0$ and close to zero otherwise. If the laplace value increases after adding a new rule, this rule should be kept otherwise discarded.

$$h_{Lap} = \frac{|\oplus|+1}{|\oplus|+|\ominus|+2} \tag{4}$$

It is also possible to use the entropy value from before for the selection of new rules. The rule based learner also computes the heuristic value for every possible new rule and then selects the rule with the highest value.

Low complexity rule based learners can often be more expressive than low complexity decision trees. This makes the computation less time consuming and is a main reason why we also chose a rule based learner. We wanted to see whether or not a rule based learner can keep up with a decision tree, while being less complex.

3.2.3 Support Vector Machines

The support vector machine is a binary linear classifier, which means it is usually applied to two-class problems. When more than two classes have to be classified we have to use the one-versus-all principle. Suppose we have four different classes. One-versus-all means that the classifier only decides if a given instance falls into class 1 or not. When it is not class 1 this only means it is one of the other three, and this can then be solved by again deciding if the instance falls class 2 or not and so on. This has to be done for every single class that can be classified.

The support vector machine basically works by fitting a hyperplane into all the feature vectors from the training set, so that the hyperplane separates the examples into two classes. The feature vectors have to be numerical in order for this to work. Every feature vector can then be plotted into a vector space. Figure 3.2 demonstrates how this looks. The blue line separates the one class (green) from the other (red). When supplied with a new example, the support vector machine classifies the green class when the given feature vector points to a point above the blue line and the red class otherwise.

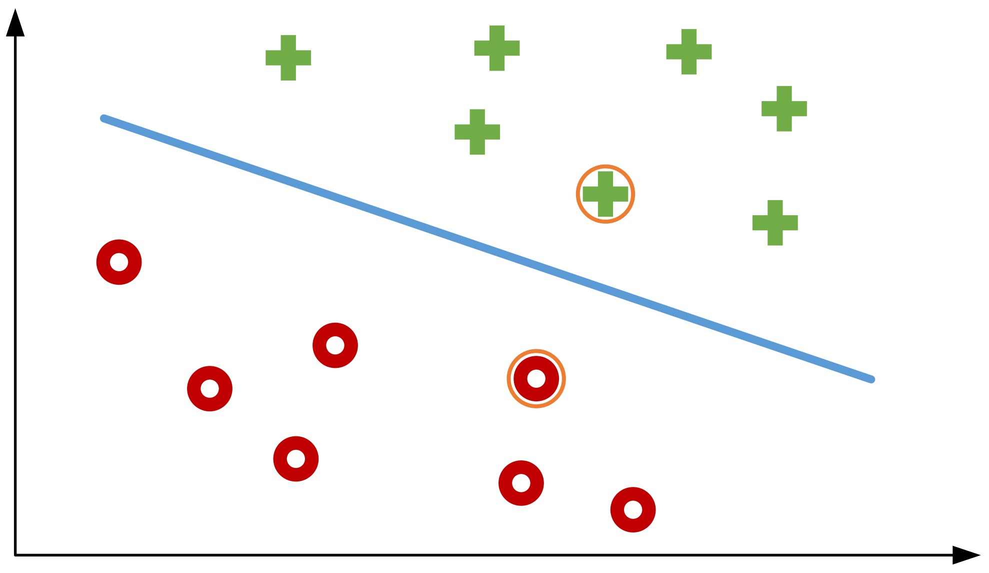


Figure 3.2: The blue line in the figure represents the linear hyperplane (or decision boundary) that separates the training examples into the two classes. The two support vectors are marked by the orange circles.

A support vector machine finds the optimal hyperplane by using the training samples that are nearest to the decision boundary. These training examples are called support vectors and the distance between the support vectors and the decision boundary is called the margin. The optimal hyperplane is found by maximizing the margin for both sides, so that the decision boundary has an equal distance to both classes. This however, is only applicable to linearly separable training data.

In case the training data is not linearly separable we have to use what is called the "kernel trick". The linear support vector machine uses the scalar product to calculate the margin of the decision boundary. In the kernel trick this scalar product is replaced by a nonlinear kernel function. This kernel function could be a polynomial function for example, which can be fit more precisely into the training data. Figure 3.3 shows some training data that is not linearly separable and a polynomial hyperplane that fits that data.

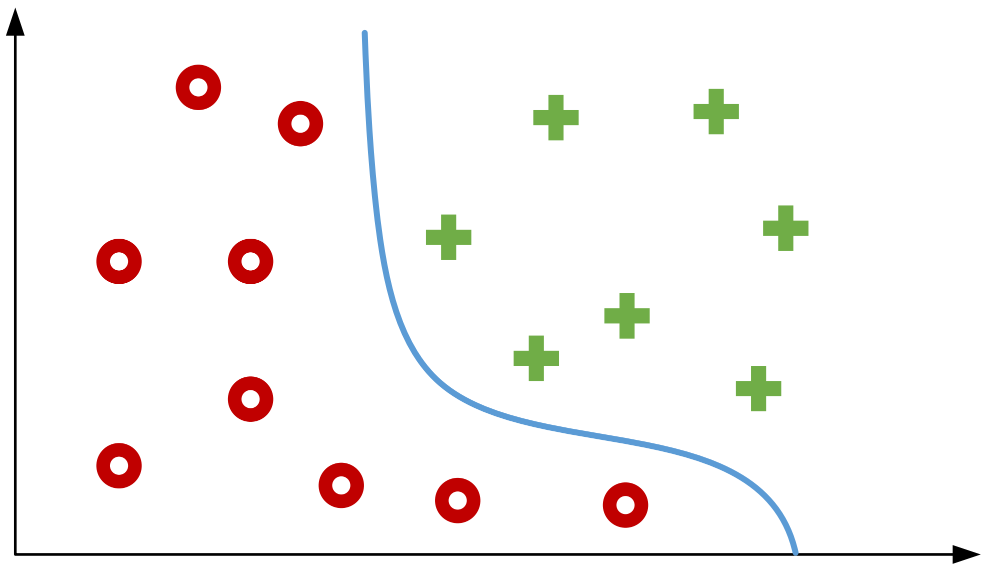


Figure 3.3: Here the training examples are not linearly separable, which is why a polynomial kernel was used to fit the hyperplane to the training examples.

3.3 Evaluation Methods

In order to evaluate a machine learning model we need a measure that reflects the performance of that model. As a basis for the evaluation the model is usually run on a test set, that is already correctly labeled. The model then classifies every instance and these classifications are then compared to the real classes. The most obvious one is the accuracy, which describes how many examples of the test set were correctly classified. Accuracy is calculated by dividing the number of correctly classified examples by the total number of examples. Another interesting value is the precision, which represents how precise the classifications of the model were. In order to explain precision we have to first explain the confusion matrix.

		Real class		
		⊕	⊖	Total
Classified as	⊕	tp	fp	$tp+fp$
	⊖	fn	tn	$fn+tn$
	Total	$tp+tn$	$fp+tn$	N

Table 3.1: A confusion matrix for a two class problem.

In a binary classification scenario the confusion matrix basically consists of four different values, tp, fp, fn, and tn as can be seen in Table 3.1. This confusion matrix must always be seen from the viewpoint of one singular class. In this case the class of interest is ⊕. The "True Positives" value (tp) is the amount of examples that were correctly classified as ⊕, whereas the "False Positives" value (fp) is the amount of examples that actually fell into class ⊕, but were classified differently. The "False Negatives" value (fn) is the amount of examples that were classified as ⊕, but actually belong to another class. The last is the "True Negatives" value, which is the amount of examples that were correctly classified as not belonging to class ⊕. When facing multi-class problems this confusion matrix fans out for every extra class.

Using the values from the confusion matrix we can now calculate the precision of a model with Equation 5 [6]. The precision indicates how precise the classification was concerning one class, in this case ⊕. In other words, from all examples that were classified as ⊕, how many actually belonged to ⊕. The difference to the accuracy is that only one class is of interest, instead of all classes with accuracy.

$$P = \frac{tp}{tp+fp} \tag{5}$$

Another interesting value is the recall. The recall indicates how many examples of one particular class were found by the model. In other words: the amount of ⊕ class examples that the model was able to find in comparison to all available ⊕ examples that could have been found. The recall is calculated as shown in Equation 6 [6].

$$R = \frac{tp}{tp + fn} \tag{6}$$

Both values, recall and precision can be combined to one value using the harmonic mean. This value is called the F-Measure and the calculation can be seen in Equation 7 [6].

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

All values are only calculated for one specific class. When we want to make an assumption on the whole performance we need to average those values. This is done by summing up for example the precision value for every class and then dividing it by the number of classes. However, as the number of instances is not always equally distributed, we have to weigh the sum by the number of instances for that particular class. Equation 8 shows the weighted average precision value for a given test set S where i is the class and S_i is the set that only includes the class i examples.

$$\bar{P} = \sum_i \frac{|S_i|}{|S|} \cdot P(S_i) \tag{8}$$

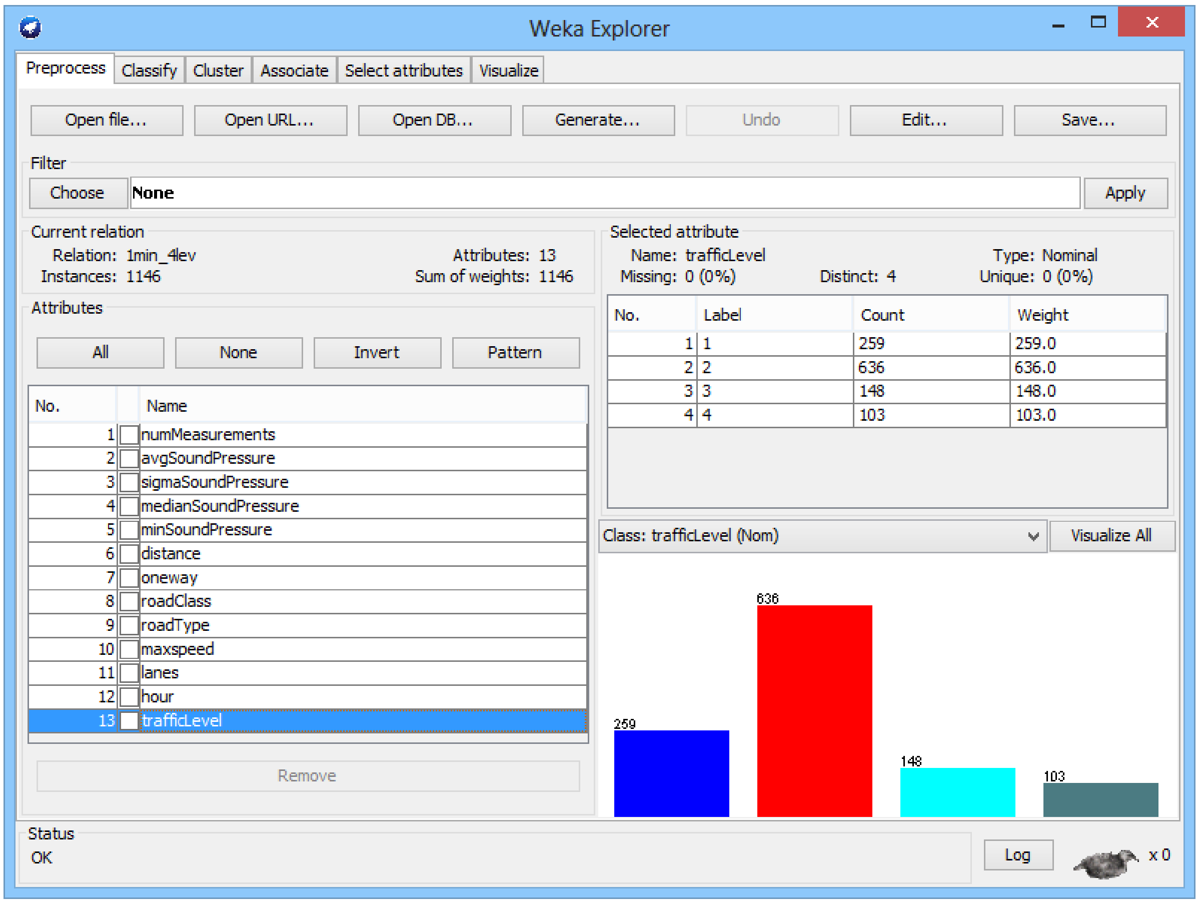
This formula can be used for the weighted average recall and f-measure by replacing the precision with the respective method.

3.4 The Weka Framwork

Within this thesis we made use of the Weka Framework [7]. The Weka framework is a machine learning suite that was developed in Java by the University of Waikato, New Zealand. It is available under the GNU General Public License.

This framework contains many machine learning algorithms that can be used directly from the graphical user interface. The user only has to supply a training set that follows the Weka specifications. All the functions can also be invoked from inside Java code importing the Weka classes into your project. Figure 3.4 shows the graphical user interface of the Weka framework after loading a training set.

The Weka framework also supports evaluation methods for the machine learning models. You can supply a specific test set yourself on which Weka will evaluate the machine learning model. In case there is no specific test set you can also choose the cross validation method. The cross validation method uses a part of the training set as the test set, i.e., it divides the training set into n equal parts and then takes one of the n parts as the test set while learning on the other $n-1$. This is done until each of the n parts was used once as the test set. Any one of the n parts is used once as the test set, which means that the machine learning has to be done n times. This is called n-fold cross validation.



Filter: None

Current relation
Relation: 1min_4lev
Instances: 1146
Attributes: 13
Sum of weights: 1146

Selected attribute
Name: trafficLevel
Missing: 0 (0%)
Distinct: 4
Type: Nominal
Unique: 0 (0%)

No.	Label	Count	Weight
1	1	259	259.0
2	2	636	636.0
3	3	148	148.0
4	4	103	103.0

Attributes

No.	Name
1	numMeasurements
2	avgSoundPressure
3	sigmaSoundPressure
4	medianSoundPressure
5	minSoundPressure
6	distance
7	oneway
8	roadClass
9	roadType
10	maxspeed
11	lanes
12	hour
13	trafficLevel

Class: trafficLevel (Nom)

Status
OK

Figure 3.4: The graphical user interface of the Weka framework can be used to run machine learning algorithms on the loaded training set. The interface also gives interesting evaluation information such as the accuracy for this run.

Weka calculates the accuracy, precision, and recall for a given model and test set. When the problem is a multi class problem, the precision and recall are calculated individually for every class. Weka also gives the average precision and recall by averaging all single class precision and recall. The F-Measure is also calculated once for every class and then also averaged.

4 System Architecture

We want to give a quick overview over the system architecture that this work is based on. An idea of how the single parts work together helps to understand the whole system. The system works basically as a pipeline of different processes. Every process creates an output that is then passed over to the next one. A flow chart of the pipeline can be seen in Figure 4.1.

The system uses publicly available data as an input and converts it into a training set. This set is then used by a machine learning algorithm to create a model that can classify the traffic situation for new examples. To do so this model only needs sound pressure measurements together with the location of some roads to classify the current traffic situation.

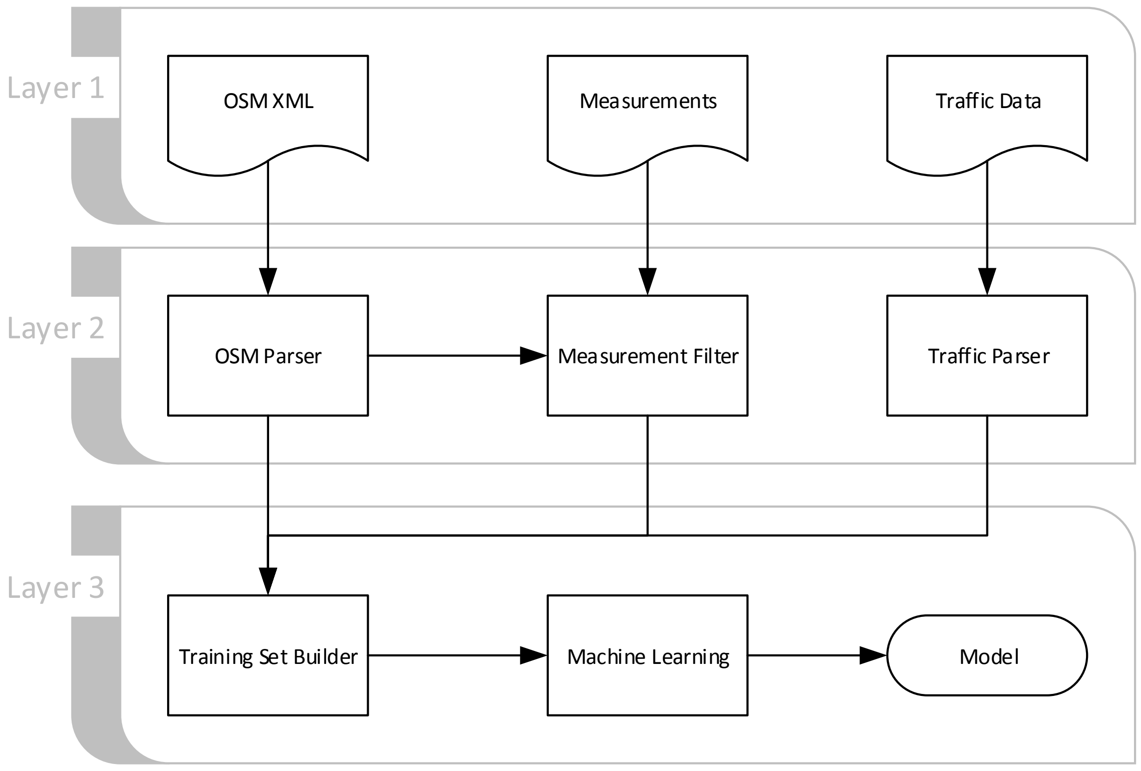


Figure 4.1: The systems architecture consists of three different layers. The first one represents the three input data sources (Open Street Map XML, the sound pressure measurements and the inductive loop data). The second level consists of the preprocessors and the final level of the machine learning.

The basic system architecture was developed by Dimitar Goshev in his Bachelor thesis [1]. The system uses three basic raw data types as input. Those have to be preprocessed in order to be usable for the next pipeline processes. Figure 4.1 can be split into three different layers. The first layer represents the raw data input and the data sources. Within this layer there is no calculation, it only includes the data input. The second layer consists of the three preprocessing units, which convert the data into a usable format. The third and final layer is the machine learning part of the system. The output of the machine learning part is the traffic level classifier.

4.1 Data Sources

We want to start by giving an overview of the different data sources we used to gather the input data for the pipeline. The data sources represent the first layer of the system architecture in Figure 4.1. We will explain the three data inputs from layer 1 in order from left to right, starting with the “OSM XML” input.

4.1.1 Open Street Map

Open Street Map (OSM) is a free project in which people around the globe created a map of the world by using collaboratively collected geo data. Everyone can contribute to this project by sending his geo data to the OSM servers in order to make the maps better. As of May 2013 OSM has over 1.1 million users [8].

OSM maps can be exported into an XML based file. For this work we exported a map of Darmstadt from the OSM website. The resulting XML file is used as the input for the OSM Parser which will be explained later in this section.

OSM does not only offer pure road maps, but also gives a lot of interesting information. It includes for example the outline of every building in Darmstadt (this can be seen in Figure 4.2). Furthermore, you can get information about every road, such as the number of lanes or the maximum speed. All this information is extracted and used later on in this work.

4.1.2 Sound Pressure Measurements

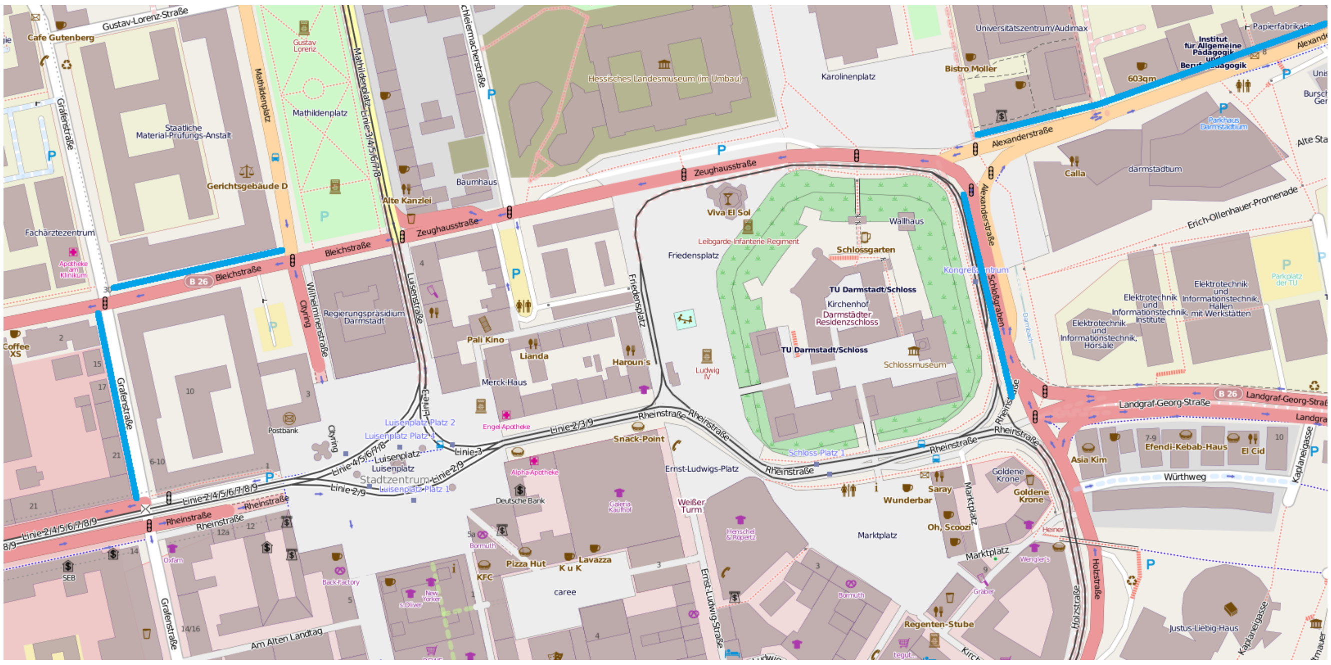


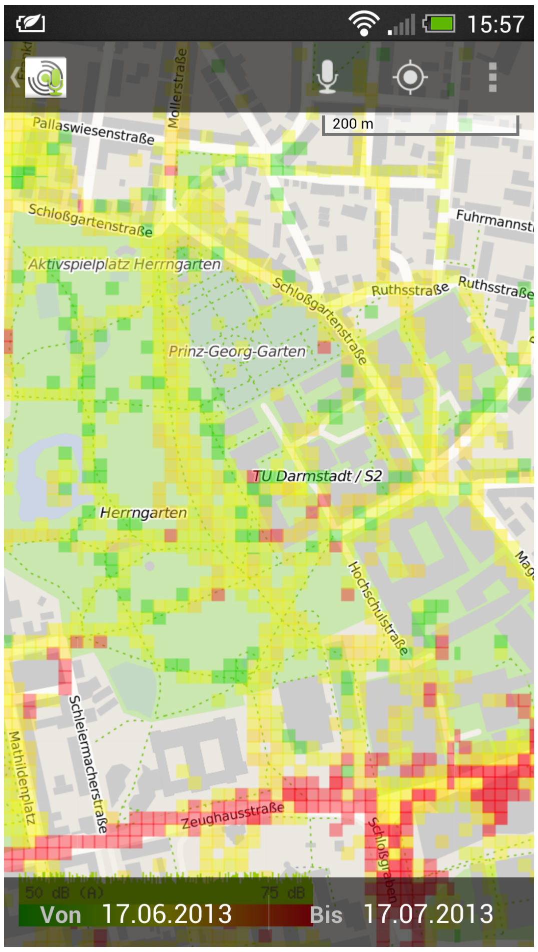
Figure 4.2: The four roads on which the sound pressure measurements were collected between the 12th and the 14th of March 2013. They are marked by the blue lines.

The sound pressure measurements we used for this work are openly available through the da_sense platform. da_sense is a data platform for mobile sensors. It was created by the Telecooperation Group of the Technische Universität Darmstadt. It uses different sensors to gather data such as the noise pollution, temperature, luminance or humidity in Darmstadt. All collected data is available for the public [9].

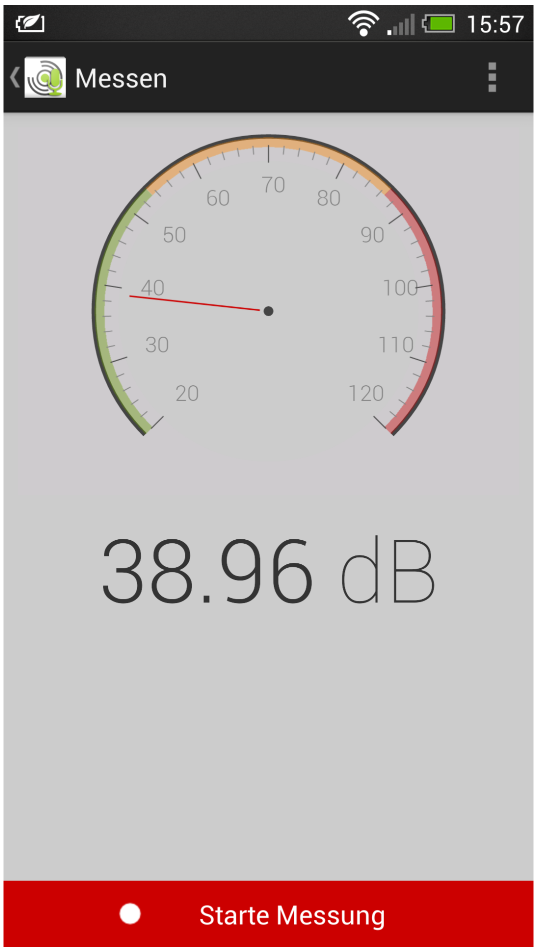
Temperature, luminance and humidity are measured by using location aware sensors on Darmstadt's trams. These sensors were installed on top of the trams and use a solar cell to recharge their battery. Using solar cells makes these sensors really durable. They can remain on the top of the trams for a long period of time. The sensors save the measured data on an internal SD card and transmit it to a main station whenever they are in range long enough. This main station is connected to the Internet via GSM (Global System for Mobile Communications) and sends the gathered data to the da_sense database. This data is then available on the da_sense website. More information on this application can be found in Marcel Wlotzka's bachelor thesis [10].

The important data from the da_sense platform for our purposes is the sound pressure. It is captured using the microphone of modern smartphones. The smartphones have to run an application called "Noisemap App". The "Noisemap App" is a mobile application which was developed by the TU Darmstadt to measure noise pollution [11]. It is currently available as an android application, but other versions including an iOS app are in development. The app measures the sound pressure over a period of 5 seconds and then averages the results so that one sound pressure value is assigned to every 5 second interval. This measurement is then combined with its GPS location and temporarily stored on the mobile phones memory. Every measurement is stored inside a database together with its GPS location, the mobile phone's device ID, the user ID and a timestamp. This database is the main data source for the measurement filtering which will be described later in this section.

Two screenshots of the app can be seen in Figure 4.3, one which shows the heat map of all measurements and another one that shows the measurement creation interface.



(a) Map view of the measurements.



(b) Interface to take new measurements.

Figure 4.3: Two screenshots of the Noisemap app.

During the early work on this thesis we encountered the problem, that many measurements from the Noisemap app were not usable. Many of them were taken by users of the platform which had no intention of capturing traffic noise. This resulted in many measurements that were taken while driving a car or sitting in a park far away from the road noise. Others were just taken at roads that were not equipped with inductive loops, which means that we cannot use these measurements for the creation of the training set, as we were unable to calculate a traffic level. Not all of them could be used and had to be filtered out. After filtering out all the unwanted data only very few remained. Of course, this was not a good data foundation, which is why we decided to do the measurements by ourself. Throughout one week a four people captured the road traffic noise at 4 different roads (Figure 4.2) in Darmstadt per day. We measured within two time slots every day. One from 10pm till 12pm and another one from 13pm till 15pm. Within that week we were able to create a sample set of over 25.000 measurements.

The four roads range from small over medium sized to big roads with up to 4 lanes. These four roads can be seen in Figure 4.2 and they will be used as the example roads throughout this thesis. Furthermore, all these roads provide perfect coverage by the inductive loops, which means that we can track every single car that drives on one of these roads. This leads to a high accuracy in the number of cars. This is not always the case for every road, many roads are only covered on the main lanes, but not the turning lanes for example. Those roads can not be used, because we cannot compute the correct number of cars. Table 4.1 shows the names of those four roads together with the corresponding amount of lanes.

These roads were chosen because of their close proximity to the Technische Universität Darmstadt. We plan to expand the number of roads in the future.

Road	Lanes
Alexanderstraße	2
Grafenstraße	2
Schloßgraben	3
Bleichstraße	4

Table 4.1: The four measured roads including their corresponding amount of lanes.

4.1.3 Traffic Data

The traffic data we use also comes from the da_sense platform. This data is collected via inductive loops. Inductive loops are the state of the art method of measuring the amount of cars on a road. They can recognize metal that hovers over them [12]. Most of the big intersections in Darmstadt are equipped with inductive loops. They are mainly used for traffic light control, so that the traffic light intervals can be adjusted to the actual traffic condition.

A total of 171 intersections in Darmstadt are equipped with inductive loops, but not every road is completely observable. In order to count the total number of cars that travel on a road, every single lane of a road needs to be equipped with an inductive loop. However, this is not the case at all of the 171 intersections. After manual checking of every intersection only 16 roads were found to be equipped with inductive loops on every lane. This means we can count the total number of cars that traveled on one of these 16 roads within a given 15 minute time interval very accurately, as no cars can disappear through a lane that is not equipped. The time interval is 15 minutes long, because the inductive loops in Darmstadt were configured this way. They only save the data once every 15 minutes. This is to be changed in the near future. At the time of this writing the inductive loops are actually using a 1 minute time interval. However, as the switch from 15 minute intervals to one minute intervals was made after we finished our research, one minute time intervals were not used within this work.

Inductive loops can produce a very accurate census of cars, but the sheer amount of cars is not the only interesting value. They can also count the amount of time that a car hovered over them. This can be used to calculate a utilization value for one loop. That is the amount of time in which the inductive loop was in an active state (a car is standing on top of it). The inductive loops in Darmstadt produce two different values that are used within this thesis: the amount of cars and the utilization factor.

1. Cars (c): the amount of cars that passed over the inductive loop within a 15 minute time interval.
2. Utilization (u): the percentage of how long the sensor was utilized within 15 minutes, i.e., the amount of time a car stood on top of the sensor divided by 15 times 100.

As mentioned earlier these values are stored into a database once every 15 minutes, after that the counters are reset. Exports from this database are available in form of “CSV” files. Within this database Darmstadt is split into the northern and the southern roads, which is why there are two export files for each 15 minute interval. One for the northern part and one for the southern one. This data is available for download from the da_sense platform [13]. Within the CSV files we can find the single values (cars and utilization) for every single inductive loop sensor in Darmstadt, either the northern or southern part. These files are parsed and imported into a PostgreSQL database for later use.

Throughout this thesis we will use c as the number of cars and u as the utilization factor. These values are always depending on the specific road and time interval. Furthermore, we use cpl as the amount of cars per lane and ups as the utilization per sensor, which are normalized values of c and u. The actual calculation will be explained later.

We tested the accuracy of the inductive loops by manually counting the number of cars that passed over the loops in one 15 minute time interval. The average amount counted by two people on that road and the actual inductive loop count only differed by 1.

4.2 Preprocessing

The second layer of the pipeline is the preprocessing. It consists of three processes that will be explained here, which take the raw data input from the data sources and transform them into data formats that can be used by the other processes. We again explain every process in order going from left to right and starting with the Open Street Map Parser.

4.2.1 Open Street Map Parser

The OSM Parser parses the exported OSM map of Darmstadt and creates an internal representation of Darmstadt and all its roads inside of a “PostGIS” [14] database. PostGIS’s functionality is superior to the basic PostgreSQL functionality, because it has support for geo spatial data.

At first the OSM Parser generates geometries for every information in the XML map file. Geometries are basically multi polygons. This includes roads, buildings, road segments, and ways. All these geometries are stored inside of a database. The advantage of using a PostGIS database is that we can use the geospatial functions it provides. PostgreSQL alone, though having geometry types on its own, is not capable of handling spatial data.

The OSM Parser also extracts interesting information about every road from the XML file. For example it extracts the number of lanes a road has, the maximum traveling speed, whether the road is one way or not or even a road type. The road type can be for example residential, tertiary, living street or primary. This information is later used to generate features for the machine learning.

In the next step the OSM Parser creates so called “sample areas”. Every sample area is also represented by a geometry object in the PostGIS database. They represent the areas around a road in which measurements are of interest. That means every measurement which is taken within one of these areas can be used for training the machine learning algorithm.

The most successful approach was to use the exact shape of the road as a sample area and then increase the outer margin to include the sidewalk of those roads. The OSM Parser was developed by Dimitar Goshev and was used within this thesis without any alterations. The exact mechanics can be looked up in [1].

A distortion is created by measurements that are taken near intersections, because noise from the one road mixes with the noise from the other road when one gets to close to an intersection. This means that there is a certain area near intersections that has to be filtered out. This was done by adding a minimum distance that every sample area has to have from the connected intersections. This distance can be seen in Figure 4.4 as indicated by the white arrow in the lower left. The minimum distance from intersections can be set manually in the software. Within this thesis we use a constant value of 15 meters for that distance, because within 15 meters the smartphones are still able to capture the sound of the road. When a measurement is taken more than 15 meters away from a road it is likely to be tainted by background noise.

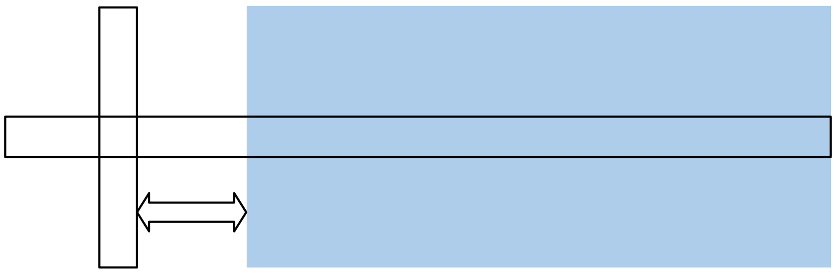


Figure 4.4: This image shows the sample area (blue) in which measurements are relevant. The distance from intersections that is necessary to stop the sounds from mixing can bee seen on the lower left side of the image (white arrow).

Measurements that are taken inside of buildings also distort the machine learning as they are not or very slightly affected by the traffic noise. This is why such measurements also have to be filtered out. The OSM data provides geometries for most buildings in Darmstadt. This data can be used to update the sample area by removing the building geometry from the sample area where they are overlapping. This is done by building the complement of the current sample area and all building geometries that overlap that sample area. You can see the updated sample areas for the intersection of Alexanderstraße and Schlossgraben in Figure 4.5.

Letting SA be the sample area and $B_1, B_2, ...B_n$ the buildings that overlap SA the formula shown in Equation 9 calculates the updated sample area, see [1] for more information.

$$SA_{new} = SA_{old} \setminus (B_1 \cup B_2 \cup ... \cup B_n). \tag{9}$$

This was done using the PostGIS command `ST_Difference(geometry A, geometry B)`, which takes two geometries as input and returns a new geometry that represents the part of A that does not intersect with B. The building geometries that intersect with the sample area can be found with the `ST_Intersects` command and checking whether a building is overlapping the sample area or not.

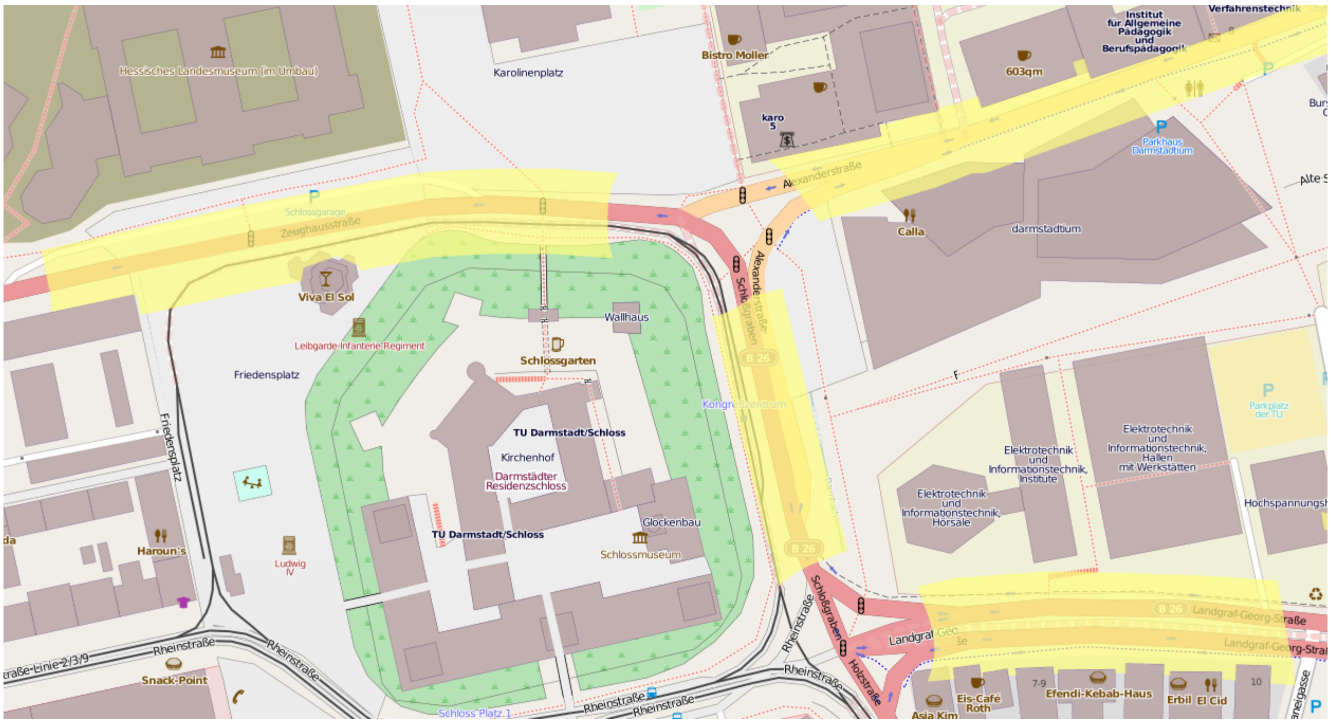


Figure 4.5: This image shows the sample areas (yellow) at the intersection of Alexanderstraße and Schlossgraben. You can see the exclusion of buildings and the minimum distance from near intersections. Furthermore, you can see the exclusion of building geometries (e.g., in the lower right corner).

4.2.2 Traffic Parser

The Traffic Parser gets the two CSV files from the da_sense platform [13] as input and parses them. These files only contain the counts for every single sensor, not for a single road. This is why the total amount of cars has to be calculated by summing up all the single values from all the sensors at one road. Figure 4.6 shows the intersection of Schloßgraben and Alexanderstraße. To get the total amount of cars, that drove on Alexanderstraße we have to add up the sensors "DK53", "DK54" and "DK72". Those three sensors are sufficient to completely cover all lanes from Alexanderstraße. The three sensors are marked by the green circles in Figure 4.6.

The same is true for the utilization of that road, except that the utilization is a percent value. Just adding up the three sensors is not sufficient; we also have to divide that sum by the number of sensors to get the utilization of the entire road. Without this step we would just add up the utilization values for the single sensors, which could lead to percent values over 100%, which is not desirable.

After parsing the two files and summing up the c and u values, we also calculate the cpl and ups values. In [1] the parser worked a little differently. It did not calculate the utilization per sensor but the utilization per lane. Such an approach is unfavorable because the number of lanes is not always equal to the number of sensors. Turning lanes are often equipped with sensors as well, but these do not count as actual lanes in the OSM data. This is why we chose to alternate the mechanics of the parser at this point and calculate the utilization per sensor instead.

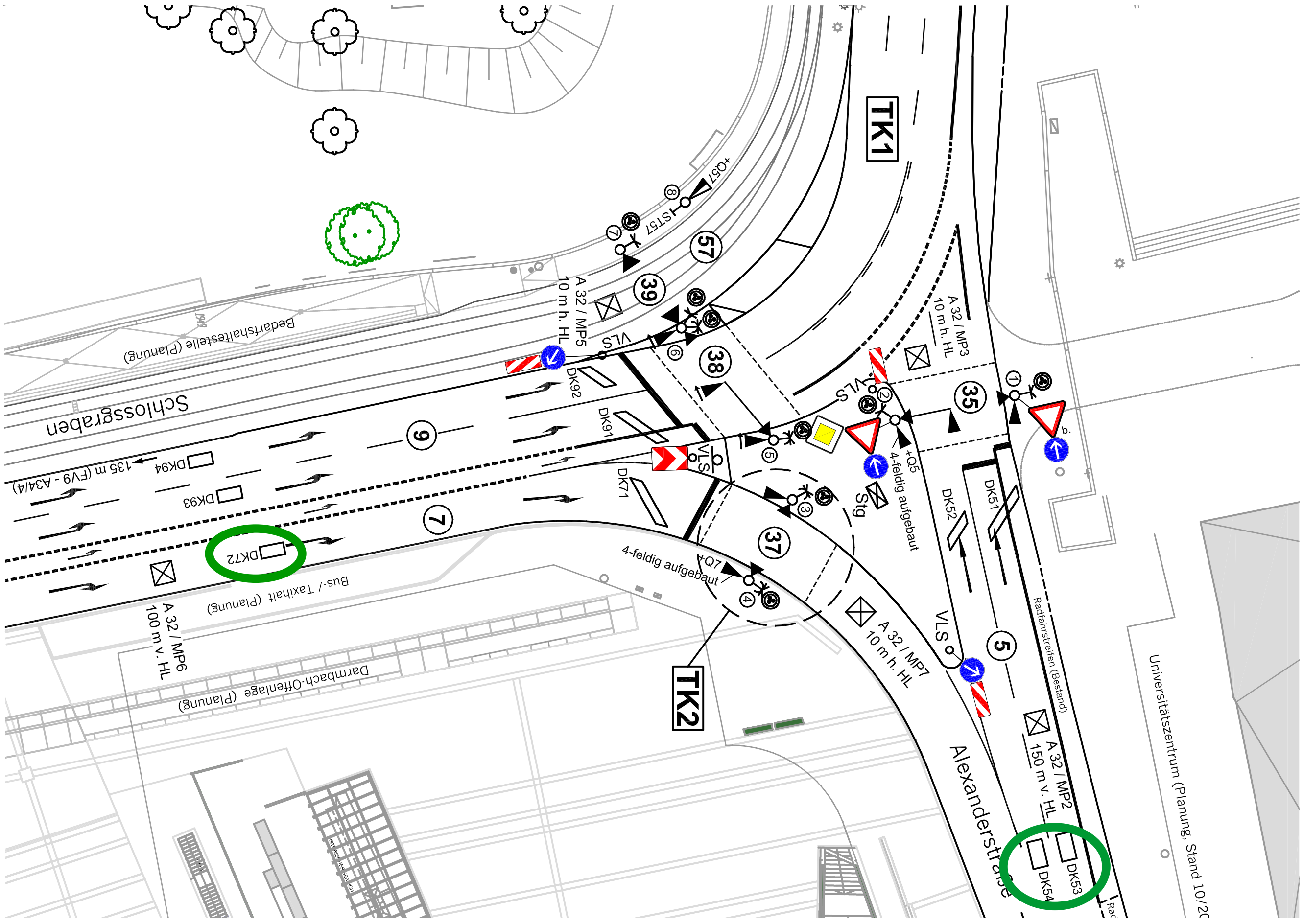


Figure 4.6: This image indicates the three inductive loops which are used for calculating c and u for the Alexanderstraße.

Every one of the 16 roads from Section 1 has a unique road ID within the PostgreSQL database. With this we can assign c, u, cpl and ups to a single road ID and insert this entry into the database. The last value we need to add is a timestamp. This timestamp can easily be calculated from the filename of the CSV file. Those files use the pattern "yyyymmdd_hhmm_[south|north].CSV". It is important that this timestamp always marks the end of the 15 minute time interval and not the beginning. This has to be taken into account when calculating the actual interval. Nevertheless the date time combination is translated into a millisecond unix timestamp and added to the entry from before. As the OSM data did not provide any information about the number of inductive loop sensors on the road, this information had to be inserted by hand for every one of the 16 roads.

Now the database traffic data table is set up and ready to be used by the later processes. Mostly the Training Set Builder makes use of the traffic data. It uses it for the classification of the traffic condition of the training samples.

4.2.3 Measurement Filter

The Measurement Filter is used to filter out all unwanted measurements from the ones that were imported from the da_sense database into a database table called `measurements`. It uses the sample areas that were generated by the OSM Parser to decide whether a measurement is of interest or not.

We start by calculating the distance of a given measurement location to every one of the 16 sample areas. The PostGIS command `ST_distance(geometry A, geometry B)`, which returns the minimum distance between two geometries, is very useful for such calculations. Whenever a measurement lies inside one of the sample areas this distance is equal to 0. When all of the 16 distances are greater than 0, the measurement is neglected. We can also directly determine at which road this measurement was taken. The road corresponding to the sample area which the measurement lies in is the road on which traffic noise was measured. This information is also stored inside the database. At this point we made another alteration to the original method from [1]. In [1] the software would filter the database everytime it runs. We added an extra step in which we filter the measurements and store them in a new database table called filtered_measurements. Every measurement that lies inside one of the sample areas is then inserted into this table together with the corresponding timestamp, ID, road ID and of course the sound pressure value. We also add the actual distance from the road to the entry. The distance from the road is significant, because the further you are away from the road the lower the traffic noise gets. This can be used as a feature for the machine learning. Adding this extra step saves runtime of the actual machine learning algorithm as the database that has to be scanned (filtered_measurements) is now much smaller than before (measurements), because it only contains the measurements that lie inside one of the sample areas.

In summary, the filtered_measurements table contains one entry for every measurement that was taken inside one of the 16 sample areas. These entries consist of an ID, timestamp, road ID, the sound pressure value and the distance from the road.

4.3 Training Set Builder

The Training Set Builder is the most important process of the pipeline. The output of the OSM Parser, the Measurement Filter and the Traffic Parser all flow together at this point. The Training Set Builder generates the training set which is used by the machine learning algorithm to create the model. All the available information is gathered at this point and combined into feature vectors.

The training set consists of the following features, where S is a list of sound pressure measurements.

- Number of measurements ($|S|$)
- Average sound pressure ($\bar{S}$)
- Standard deviation of sound pressure ($\sigma(S)$)
- Median of sound pressure measurements ($\tilde{S}$)
- Minimum of sound pressure measurements ($\min_S$)
- Distance from the road
- Boolean, whether the road is oneway or not.
- The road class
- The road Type (e.g., motorway, trunk, primary)
- Maximum speed of the road
- The number of lanes
- The hour when the measurement was taken

These features are basically the features that were used in [1]. The most important feature in [1] was the average sound pressure, because it was the only feature that actually depended on the traffic noise. We added more features, that are based on the sound pressure measurements to the feature vector. For example the standard deviation is very interesting. It can indicate the actual traffic condition. When the traffic situation is not congested and many cars are moving, the standard deviation should be very high, as the sound pressure goes up first and then down when a car is passing the smartphone. When the traffic situation is congested, on the other hand, the cars are not moving, hence generating a more consistent sound pressure. This would cause a very low standard deviation. The minimum sound pressure was chosen to indicate if there was a certain time within the measuring period in which no cars passed by the smartphone. When this is true the minimum sound pressure should be around 30 dB. Consistent traffic would not allow such a low minimum sound pressure. We also added the median sound pressure to handle outliers. The hour of the measurement is another feature we added. Traffic volume changes depending on the current daytime. The rush-hour starts around 6am for example so naturally the traffic level is higher at this time. This feature can easily be calculated out of the measurement timestamp. The last new feature we added is the road class. We discovered that there are different kinds of roads, some are bigger, some are smaller and so on. The road class will be explained in detail in Section 5.2.

The training set builder starts by extracting all measurements from the filtered_measurements database table. Every measurement has a timestamp, which can be used together with the road ID to find the correct traffic data interval. Not every measurement results in one training sample in the training set. At first we used every measurement that was taken at one road and one 15 minute time interval. This however, led to a very low amount of sample instances. To increase the amount of instances in the training set we split every 15 minute interval into 15 one minute intervals. Every one minute interval contained only the measurements that were created within this minute. The traffic data for each of those one minute intervals is the 15 minute interval traffic data that they lie in. Using this method we were able to increase the number of instances in the training set by a factor of 15. This is beneficial for the machine learning, as more instances often lead to a better machine learning model. The more instances the machine learning model gets, the more it can adapt to the given problem.

Now the training set builder calculates the values for the features. The number of measurements represents the amount of measurements that are available in this specific 1 minute time interval at a specific road. Sound pressure measurements are measured in decibel, which is a logarithmic scale. This means that the average sound pressure has to be calculated by using the "equivalent continuous sound level", which can be seen in Equation 10, where n is the number of measurements, R a particular road, T is a time interval, and $S_{T,R}^{(i)}$ is the i^{th} measurement within T on road R.

$$\bar{S}_{T,R} = 10 \cdot \log_{10} \cdot \left(\frac{1}{n} \sum_{i=1}^{n} 10^{0.1 \cdot S_{T,R}^{(i)}} \right) \tag{10}$$

The standard deviation is calculated by taking the square root of the sound pressure measurement variance. The variance is defined as shown in Equation 11 [15], [16], hence the standard deviation is given by $\sigma(S_{T,R})$.

$$\sigma^2(S_{T,R}) = \frac{1}{n} \sum_{i=1}^{n} \left(S_{T,R}^{(i)} - \bar{S}_{T,R} \right)^2 \tag{11}$$

Equation 12 is used for the calculation of the median of the sound pressure measurements [15]. The Formula requires a sorted list of all the measurements and uses $S_{T,R}^{(i)}$ as the i^{th} measurement in that list, T is the timestamp marking the beginning of the one minute interval and R is the road ID.

$$\tilde{S}_{T,R} = \begin{cases} S_{T,R}^{((n+1)/2)} & \text{if } n \text{ odd} \\ \frac{1}{2}\left(S_{T,R}^{(n/2)} + S_{T,R}^{((n/2)-1)} \right) & \text{if } n \text{ even} \end{cases} \tag{12}$$

The minimum sound pressure is just the first item of the sorted sound pressure measurement list. Using the OSM data we can calculate the oneway, road type, maximum speed and number of lanes features. All this information is available through the OSM Parser. However, we discovered that the number of lanes is not accurate in some cases. This is due to the fact that turning lanes are often not counted as actual lanes, although they are part of the complete road. This is true for the Grafenstraße in our case. Lastly the distance from the road is calculated again by using `ST_distance(geometry, geometry)` with the first geometry being the actual road and the second one the position of the sound pressure measurement.

Supervised machine learning algorithms require an already labeled training set. The last feature in every training sample vector is labeled with the expected classification. In our case this is the traffic level. The Training Set Builder has to calculate the traffic level that corresponds to the actual traffic condition at that time. It uses the data from the induction loops to do so, i.e., it uses cpl and ups to calculate the traffic level. First it finds the corresponding 15 minute time interval in which the one minute interval is included and extracts the traffic information from the database. It then calculates the traffic level using an algorithm that will be explained in detail in Section 5.

```
@relation exampleArffFile

@attribute numMeasurements numeric
@attribute avgSoundPressure numeric
@attribute sigmaSoundPressure numeric
@attribute medianSoundPressure numeric
@attribute minSoundPressure numeric
@attribute distance numeric
@attribute oneway {true, false}
@attribute roadClass {1,2,3}
@attribute roadType {motorway, trunk, primary, secondary, tertiary, residential, livingstreet}
@attribute maxspeed numeric
@attribute lanes {1,2,3,4,5}
@attribute hour numeric
@attribute trafficLevel {1,2,3,4}

@data
441,86.874927,16.216025,83.7436,43.1513,2.224172,true,2,primary,50,5,11,3
35,93.079668,2.266088,92.7251,86.0307,7.035779,true,2,primary,50,5,14,2
60,89.730386,11.148523,88.99595,57.1165,2.20194,true,2,primary,50,5,11,2
59,98.140933,33.02859,91.5476,38.8452,3.055015,true,2,primary,50,5,11,2
351,87.492878,15.345098,85.0404,43.1751,2.603064,true,2,primary,50,5,10,3
207,92.689272,40.004851,49.164,35.5038,2.507913,true,2,primary,50,5,10,2
```

Listing 4.1: Example of an ARFF file.

In the end the Training Set Builder outputs an "ARFF" (Attribute-Relation File Format) file, which is compatible with the used Weka framework [7]. You can see an example of such a file in Listing 4.1. In the first line we can see `@relation` attribute, which represents the name of the training set. The following lines that start with `@attribute` are the definition of the feature vector. One `@attribute` stands for one feature inside the feature vector. The first parameter after the `attribute` is the name of the feature and the second parameter is the feature type. This can be `numeric` or a list of possible values. The `trafficLevel` for example can only be one of the values 1, 2, 3 or 4. The last `@attribute` is the expected value, that the machine learning algorithm should classify, when getting the other features as input.

5 Traffic Levels

The main part of this work was the actual calculation of a traffic level from the traffic data. In this section we will present the different approaches we worked out. We started with a simple approach and refined this one in every step along the way.

To fully understand our approaches we will give a short introduction to the values and variables that were used to develop these approaches in the next section. After that we will present our idea of road classes. We found that every road in Darmstadt can be assigned to a specific road class by using correlation between the cars and the utilization on that road. Then we will dive into the actual calculation of the traffic levels.

5.1 Overview

We want to start this section with a quick explanation of the different variables and values we use. The two variables which can directly be parsed from the CSV files are c (amount of cars within 15 minutes) and u (utilization in seconds within 15 minutes), but those are not representative, because they depend on the type of the road and the number of sensors a road has. This means that a road with more lanes has a bigger capacity and hence a higher average number of cars per 15 minutes. This is why we normalized these values.

Utilization is the percentage value of how long one or more cars stood upon an inductive loop. This value only represents the utilization of one single inductive loop sensor. When we just add up the utilization of every sensor a road has, this can lead to values over 100%. To get the average utilization value for one entire road we divide the sum of all sensors by the number of sensors which leads to

$$ups = \frac{1}{n} \cdot \sum_{i=1}^{n} u_i \tag{13}$$

where n is the number of sensors for a given road and time interval and u_i the utilization for sensor i.

For the amount of cars this is a little different. Just adding up the c for every sensor on a given road leads to the total amount of cars that drove on the road within that one 15 minute time interval. This value alone is a good indicator for the traffic on one road, but this does not take into account that roads can have multiple lanes. To normalize this dividing by the number of sensors is not a good choice as the number of lanes and the number of sensors are not always the same. Furthermore, not every big road necessarily has a higher number of sensors, which means that the number of sensors is not a good indicator for the capacity of the road. Using the number of lanes is more convenient and leads to

$$cpl = \frac{1}{l} \cdot \sum_{i=1}^{l} c_i \tag{14}$$

where l is the number of lanes for a given road and c_i the amount of cars for sensor i on that road at given time interval.

By plotting every pair of $cpl_{T,R}$ and $ups_{T,R}$ with similar T and R in a scatter plot we get to the graph shown in Figure 5.1. The graph shows the traffic data that was measured by the inductive loops between the 1st of January 2013 and the 28th of March 2013 with the $cpl_{T,R}$ on the x-axis and $ups_{T,R}$ on the y-axis.

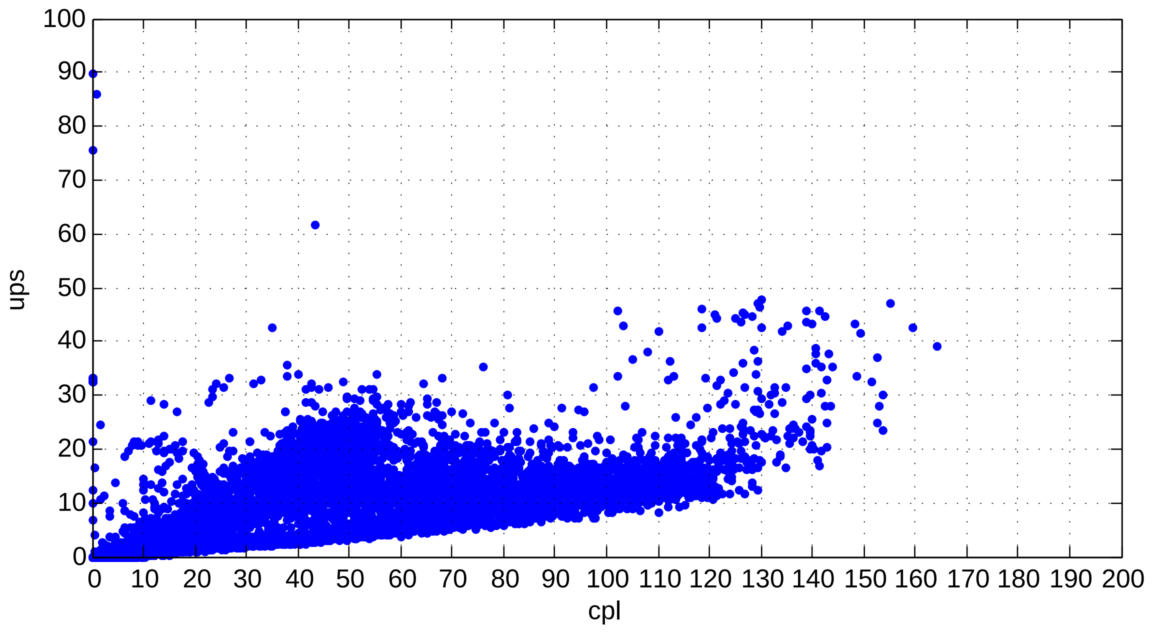
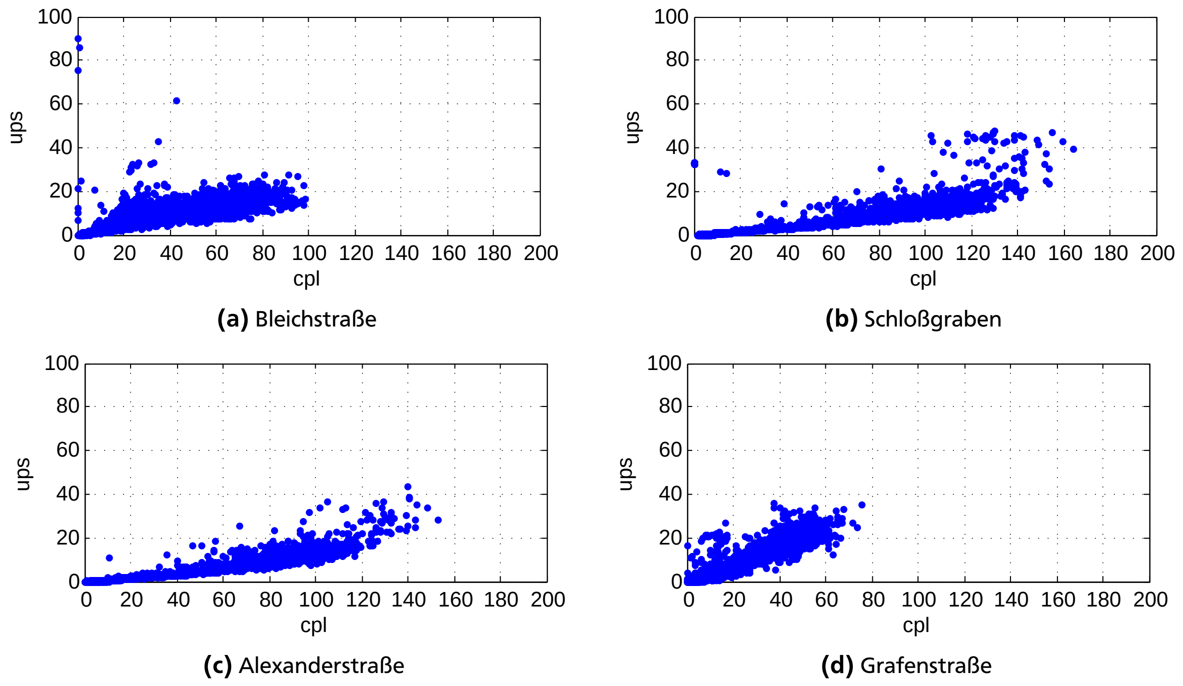


Figure 5.1: This scatter plot shows cpl and ups as they occurred between 1st of January 2013 and 28th of March 2013. All these values were generated by the inductive loops.

The distribution of the pairs looks arbitrary at first sight, but upon closer examination it reveals some interesting characteristics. When only plotting the pairs that correspond to one specific road we get the result that is shown in Figure 5.2. The graph shows all pairs for the four different roads from Figure 4.2. When examined separately the point cloud does not appear as arbitrary as in Figure 5.1. Every single road does behave almost linearly. The utilization seems to depend linearly on the amount of cars on a road. The opposite, that the amount of cars depends on the utilization, cannot be true, because without any cars on a road there can be no utilization.



(a) Bleichstraße **(b)** Schloßgraben **(c)** Alexanderstraße **(d)** Grafenstraße

Figure 5.2: Scatter plot of all cpl and ups pairs for the four roads singly.

5.2 Road Classes

Roads do not always have the same specifications, this is why we chose to use road classes to distinguish between roads that behave differently. The preceding section has shown that cpl and ups seem to have a linear relation. This means that the higher the number of cars becomes, the higher becomes the utilization on a road. Small roads can not handle a large amount of cars, i.e., a small number of cars can be sufficient to create utilization values of 25% and higher, whereas big roads with multiple lanes can handle more cars without being utilized much.

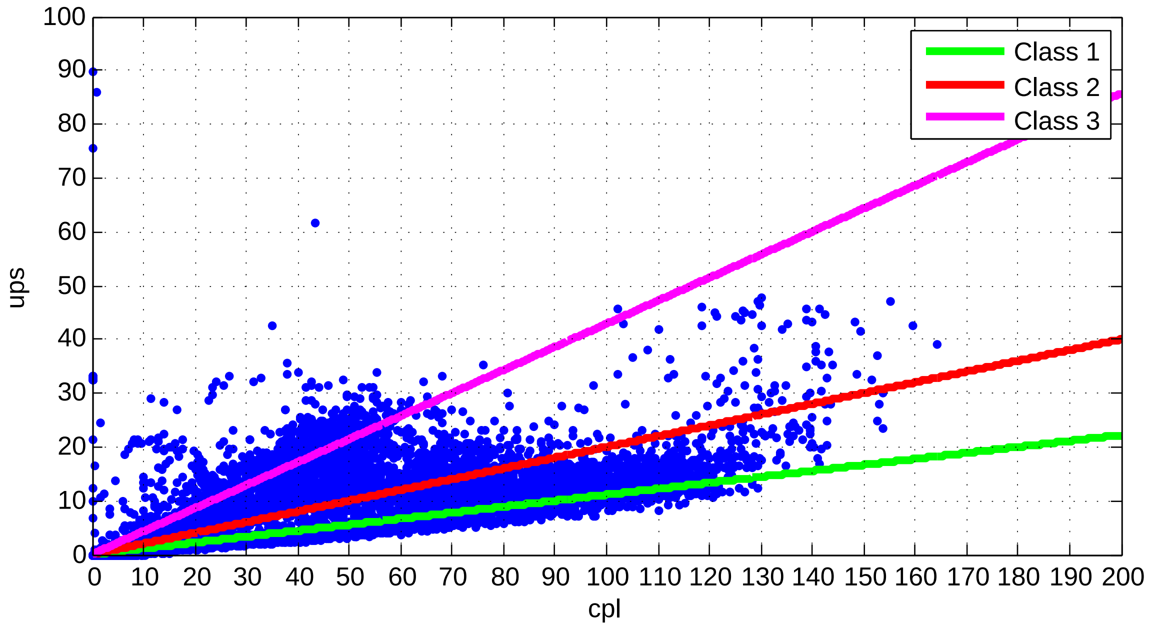


Figure 5.3: The road classes indicated by the three lines.

In Figure 5.3 the three road classes can be seen. Those three lines are estimates of the progression of the point clouds from the single roads (Figure 5.2). We tried to place these lines in the center of each roads' point cloud. The road class is defined by the slope of the linear function the point cloud of that road follows. This means we chose the road class that the point cloud is closest to. In this case the road classes are as shown in Table 5.1.

Road	Class
Schloßgraben	1
Alexanderstraße	1
Bleichstraße	2
Grafenstraße	3

Table 5.1: The road classes for every of the four roads.

The road class is an indicator for the capacity of a road. Class 1 roads, for example, are capable of handling a high amount of cars, without getting as utilized as smaller roads. Class 3 roads on the other hand have a very limited capacity, because even a low amount of cars can lead to a rather high utilization on such roads.

Road classes are very important for the traffic level classification, as we can not treat every road the same. A high cpl value might mean a high traffic level on one road, whereas it does not on another. The road class is used as a feature for the training set with the intention that the machine learning algorithm can distinguish between those types of roads.

5.3 Calculating the Traffic Level

Now we will present the different approaches we came up with to calculate a traffic level from inductive loop data. In [1] they used only one of the two variables cpl and ups to determine the traffic level. They did this by assigning a certain value to intervals. A cpl value of 80 for example would fall into the interval $(60, 100)$ which could stand for traffic level

3, whereas $(20, 60)$ would stand for traffic level 2. However, this does not take into account that the combination of the two values can give valuable information.

A high ups value together with a low cpl value for example would indicate high congestion. A high utilization of the road means that cars stood on top of the inductive loop for a long time, but not many cars passed over it. This indicates that the road was highly congested, as not many cars were able to pass the inductive loop. Just using one of the two variables would lead to a wrong conclusion here. All of the following approaches to the traffic level calculation are based on both cpl and ups.

Another question has been how many traffic levels are sufficient. In [1] traffic levels between 3 and 7 were used. Only using three traffic levels however does not represent traffic very well. We need at least one traffic level to represent a traffic jam. When using three traffic levels only two remain to represent the different states of traffic where the traffic is actually flowing. This is why we chose to use at least four traffic levels, as we can represent low, medium and high traffic as well as a traffic jam. When looking at other applications that show a traffic level such as Google Maps, we can see that they also use 4 different levels of describing the traffic situation. Furthermore, it gets more difficult to differentiate between two adjacent levels with an increased number of traffic levels.

Because there is no openly available method for calculating such a traffic level, we started with a very basic approach to the problem. We then took that approach and improved it.

5.3.1 Rectangular Traffic Levels

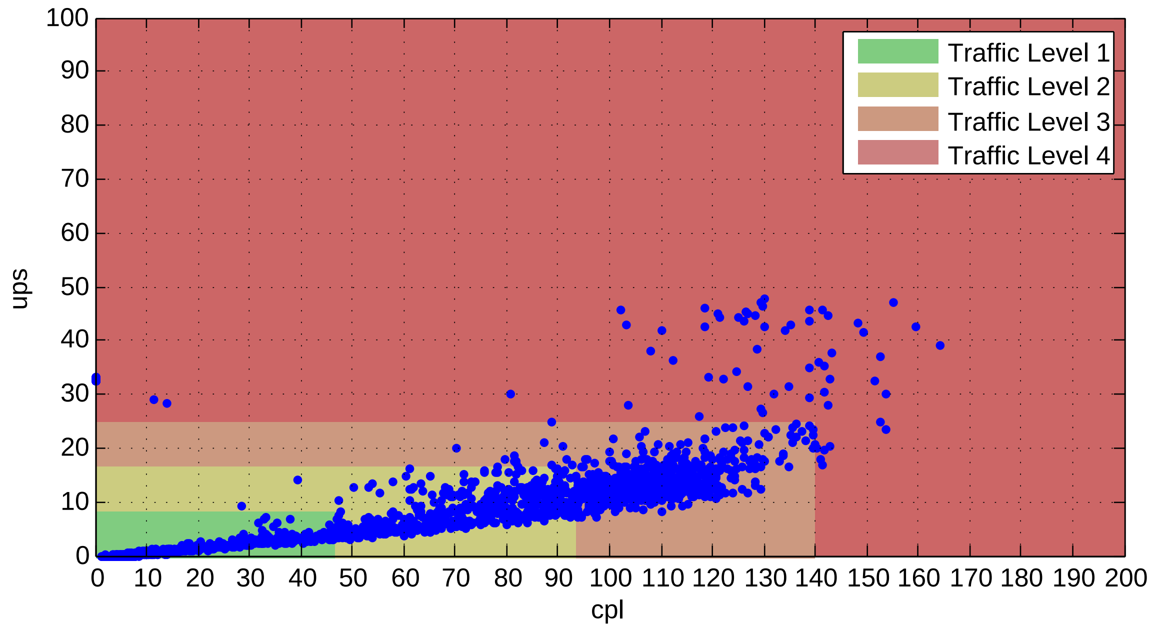


Figure 5.4: This graphic shows the rectangular traffic levels for the road Schloßgraben.

Our first approach to the problem was using rectangular shapes for determining the traffic levels. Figure 5.4 shows this approach for the road Schloßgraben. This approach is similar to the one used in [1] except that it uses both cpl and ups for the calculation.

We will introduce two helper functions that can be seen in Equation 15 which will be used to make the traffic level formulas more readable. These functions divide an interval into $(n-1)$ equal parts. When the width of the outer rectangle is 150 for example, the divisions will be at 100 and 50.

$$cpl_{max}(x) = \frac{x \cdot cpl_{max}}{n-1}, \quad ups_{max}(x) = \frac{x \cdot ups_{max}}{n-1} \tag{15}$$

Equation 16 shows the calculation of the traffic level with the rectangular approach, where n is the number of traffic levels and cpl_{max} and ups_{max} are the width and height of the $(n-1)^{th}$ traffic level. The algorithm checks whether a pair (cpl, ups) lies inside the first rectangle, if not it reiterates for the second, then for the third etc. After the $(n-1)^{th}$ rectangle has been checked without success the highest traffic level is assigned. The constants cpl_{max} and ups_{max} have to be set according to the road class.

$$TL(cpl, ups) = \begin{cases} 1 & 0 \leq cpl < cpl_{max}(1) \land 0 \leq ups < ups_{max}(1) \\ 2 & cpl_{max}(1) \leq cpl < cpl_{max}(2) \land ups_{max}(1) \leq ups < ups_{max}(2) \\ \vdots & \\ n-1 & cpl_{max}(n-2) \leq cpl < cpl_{max}(n-1) \land ups_{max}(n-2) \leq ups < ups_{max}(n-1) \\ n & \text{otherwise} \end{cases} \tag{16}$$

Using the graphs from Figure 5.2 we set the width and height of the $(n-1)^{th}$ traffic level so, that every pair that lies outside of this rectangle represents the traffic jam level. Those are pairs which have either a high cpl value or a high ups value, where the ups value has a higher weight. The constants can be seen in Table 5.2.

What is important is that ups is a better indicator for high congestion than just the pure amount of cars. A road can be so congested that not many cars are able to pass over an inductive loop. This means a low cpl value together with a medium to high ups value should be a higher traffic level. Also on smaller roads, i.e., class 1 roads a smaller cpl and ups value should be sufficient for a higher traffic level.

Road Class	max_{cpl}	max_{ups}
1	140	25
2	100	25
3	60	25

Table 5.2: The constants as they were set for the rectangular traffic levels.

This approach worked out well as a start. However, it does not use the combination of cpl and ups to calculate the traffic level. The problem was that the two values are not used together, but separately. A cpl value of 100 and an ups value of 30 would lead to the same traffic level as a cpl value of 100 and an ups value of 5, which should not be the case. The second example, though having the same amount of cars should fall into a lower traffic level, because the utilization is very low, which means that the traffic was basically flowing without interruptions.

5.3.2 Triangular Traffic Levels

Because the rectangular traffic levels did not use the combination of both variables we decided to use triangular shapes. Triangular traffic levels divide the point cloud perpendicular to the direction of the linear functions from Figure 5.3. This can be seen in Figure 5.5 for the street Schloßgraben.

To calculate whether a cpl ups pair lies inside of a traffic level we had to use the linear function for the hypotenuse of the triangle. Equation 17 shows the calculation of the hypotenuse.

$$tri_n(cpl) = ups = \frac{ups_{max}(n)}{cpl_{max}(n)} \cdot cpl + ups_{max}(n) \tag{17}$$

When ups is smaller or equal to the linear function it lies inside the triangle. This is checked for every triangle from bottom up. After the last triangle has been checked without success, the highest traffic level is assigned. The single triangles are again calculated by using the helper functions from Equation 15 and the function from Equation 17.

$$TL(cpl, ups) = \begin{cases} 1 & 0 < ups \leq tri_1(cpl) \\ 2 & tri_1(cpl) < ups \leq tri_2(cpl) \\ \vdots & \\ n-1 & tri_{n-2}(cpl) < ups \leq tri_{n-1}(cpl) \\ n & \text{otherwise} \end{cases} \tag{18}$$

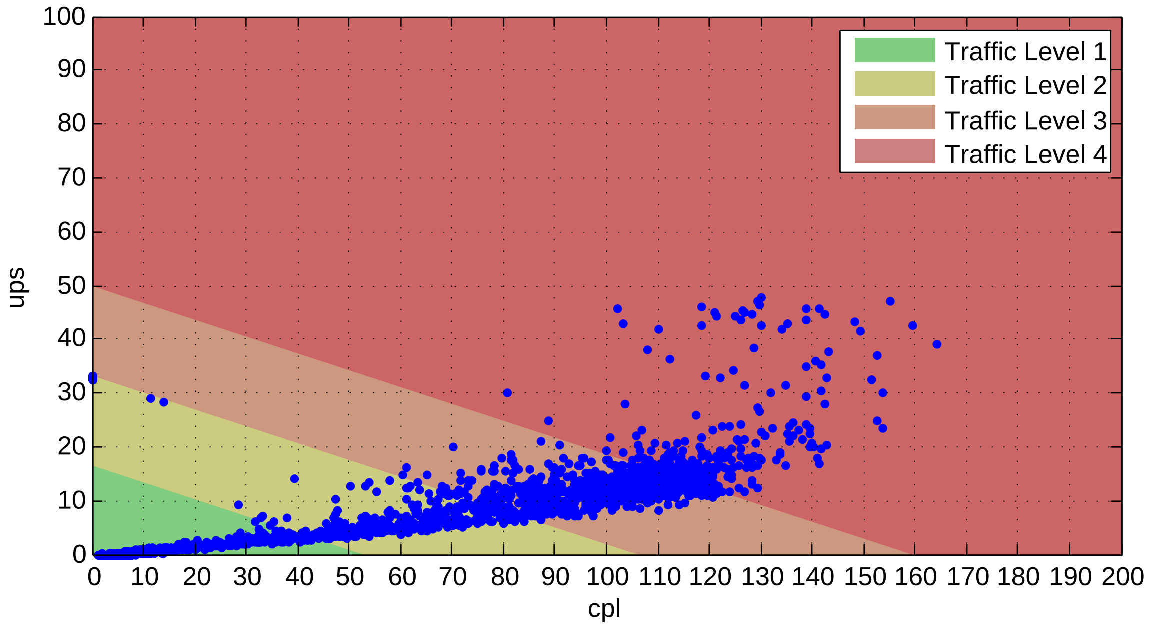


Figure 5.5: This graphic shows the triangular traffic levels for the road Schloßgraben.

The constants had to be set differently because of the different shape in comparison to the rectangular traffic levels. They were set as in Table 5.3.

Road Class	max_{cpl}	max_{ups}
1	160	50
2	120	50
3	80	50

Table 5.3: The constants as they were set for the triangular traffic levels.

This approach considered the combination of cpl and ups more, but had one major problem. The utilization was underrated. High utilization values are a good indicator for high congestion. With this approach the utilization could go up to 50% without causing the highest traffic level to be assigned. This is why we made the following alterations.

5.3.3 Elliptic Traffic Levels

This approach was a combination of both the rectangular and the triangular approach. Both previous approaches had their advantages and disadvantages. In this approach we tried to combine the pros from both. We decided to use an elliptic shape, as it gives the opportunity to rank the combination of two high cpl and ups values higher as well as just a high ups value.

The algorithm uses a formula to calculate the elliptic shape. This can be seen in Equation 19, however this formula is a simplified version, as we do not have to bother with rotations and translations of the ellipse. The center of the ellipse is the origin of the graph and the ellipse does not have to be rotated. Figure 5.6 shows the elliptic traffic levels for the road Schloßgraben. We can see that the ellipses are centered in the origin and not rotated.

$$ell_n(cpl, ups) = \frac{cpl^2}{cpl^2_{max}(n)} + \frac{ups^2}{ups^2_{max}(n)} \tag{19}$$

A point (cpl, ups) lies inside of the ellipse when $ell_n(cpl, ups) \leq 1$. The ellipses are checked bottom up whether or not the given cpl and ups value pair lies inside of it or not and the corresponding traffic level is then assigned. When the

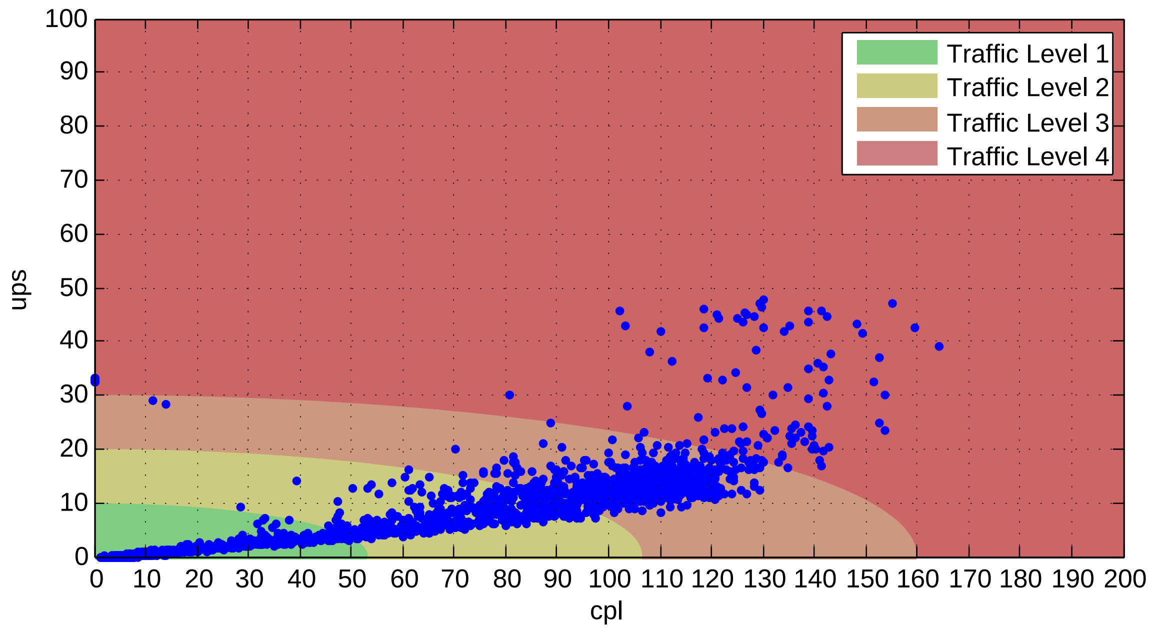


Figure 5.6: This graphic shows the elliptic traffic levels for the road Schloßgraben.

point lies outside of the $(n-1)^{th}$ ellipse the highest traffic level is assigned. Equation 20 shows the calculation of the elliptic traffic level.

$$TL(cpl,ups) = \begin{cases} 1 & ell_1(cpl,ups) \leq 1 \\ 2 & ell_1(cpl,ups) > 1 \wedge ell_2(cpl,ups) \leq 1 \\ \vdots & \\ n-1 & ell_{n-2}(cpl,ups) > 1 \wedge ell_{n-1}(cpl,ups) \leq 1 \\ n & \text{otherwise} \end{cases} \tag{20}$$

The constants were set as in Table 5.4.

Road Class	max_{cpl}	max_{ups}
1	160	30
2	120	30
3	80	30

Table 5.4: The constants as they were said for the elliptic traffic levels.

From all approaches we tried, this is the one we found best for the problem of traffic level calculation. We closely observed the traffic situation when taking the measurements and this approach represents our observations best. This is of course a subjective assessment.

5.3.4 Maximum Road Capacity

When taking a closer look at the behavior of the roads one can see that this can not be linear as stated earlier. The linear behavior is not true for a very high amount of cars. When the cars on a road are so many that the road is completely full and nothing can move, the *cpl* value will drop, whereas the *ups* value will raise significantly. Roads have a certain amount of cars they can handle and if this amount is exceeded the *cpl* value drops, but the *ups* raises. This threshold is what we call the maximum road capacity.

We implemented this into the previously mentioned approaches by adding a constant threshold at 30% *ups*, because we recognized that the linear relation between *cpl* and *ups* did not persist after the threshold was passed. This could indicate that the road is at its maximum capacity. Furthermore, the pairs above the threshold are more widespread. Our assumption is that everyone of these pairs was generated during a traffic jam situation. When a road reaches its maximum capacity, the amount of cars actually decreases, because the road is so congested that the cars cannot move. The utilization however is increasing, which is why the linear relation is not persistent after the maximum capacity is exceeded. The actual relation between cpl and ups becomes more like a spiral, the function bows back.

This is why we assign the highest traffic level whenever this threshold is exceeded, regardless of the cpl value. In Figure 5.7 we added the threshold to the elliptic approach as an extra 5th level. The highest level is always the level that represents traffic jam.

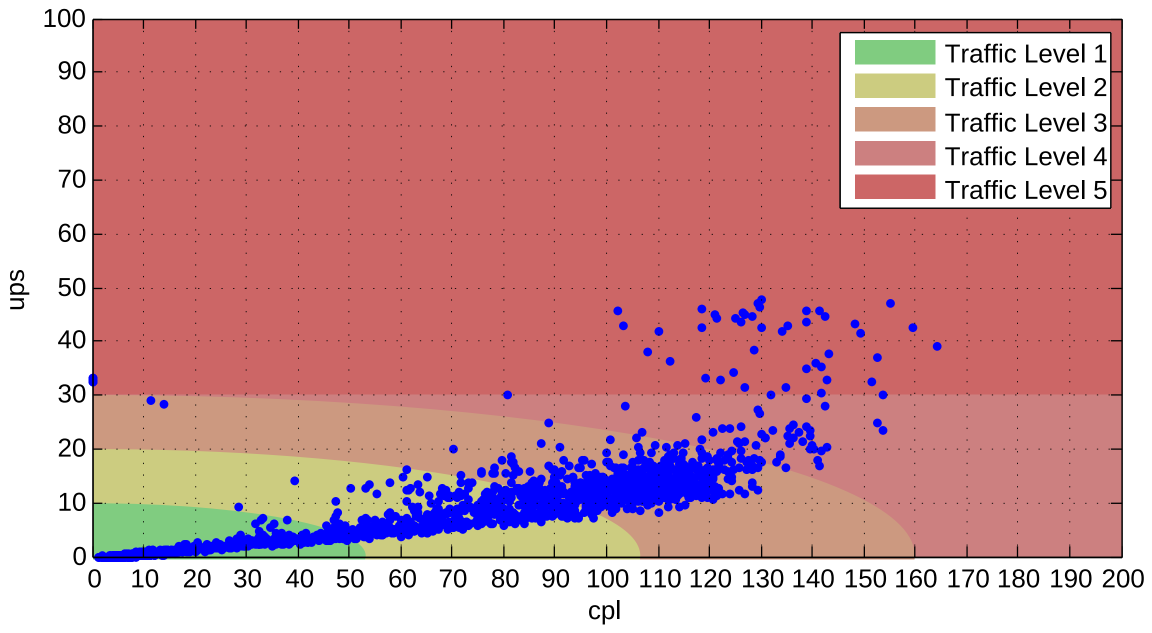


Figure 5.7: This graphic shows the threshold at 30% ups. Every point that exceeds this threshold is assigned to the highest traffic level.

6 Evaluation

The evaluation in this section was done using the Weka Framework [7], which is available under the GNU General Public License. For the support vector machine the LIBSVM package was used, which is available under BSD license [17].

Five different models have been evaluated. The difference between those models is basically the calculation of the traffic level. We also varied the time interval for one of the models, so it does not only use 1 minute time intervals, but also 5 and 15 minute time intervals. The inductive loops measure the amount of cars and the utilization once every 15 minutes. To increase the number of instances all measurements within this 15 minute interval were taken and distributed into non overlapping 1 minute intervals. This increases the amount of instances by the factor of 15. Instead of just one instance we get 15, all with the same base cpl and ups values from the 15 minute time interval.

Furthermore, we used three different machine learning algorithms for the evaluation. One decision tree algorithm, one rule based algorithm and the commonly used support vector machine. We chose these algorithms for the direct comparison because they all represent different branches of machine learning algorithms. The support vector machine was specifically chosen, because it was used in [1]. The other ones were chosen because their mathematical basis is far less complex than that of the support vector machine. It is interesting to see if a less complex approach can yield good performance as well.

A direct comparison between [1] and our approach seems natural, but is not advised. We used the basic implementation of the system architecture as a basis for our development. The resulting models behave different because the traffic level calculation has changed completely.

The algorithms were evaluated using a 10 fold cross-validation, because we had no extra test set available. Such a test set could have been created in the same way as the training set, however, we did not have the time to do so.

For the direct comparison we chose a simple baseline approach. This baseline always classifies the traffic level, that has the most instances in the training set. This means it always classifies that traffic level that is most common among all instances of the training set. When the training set would be equally distributed the accuracy of such a baseline would be $^{100}/_{n}$ where n is the number of traffic levels.

6.1 Rectangular Traffic Levels

Accuracy	82.64%
Average Precision	82.56%
Average Recall	82.64%
Average F-Measure	82.57%

(a) J48

Accuracy	79.58%
Average Precision	79.26%
Average Recall	79.58%
Average F-Measure	79.22%

(b) JRip

Accuracy	82.11%
Average Precision	82.23%
Average Recall	82.11%
Average F-Measure	80.95%

(c) LibSVM

Figure 6.1: Rectangular traffic levels with 1 minute time interval using 1146 instances. The baseline accuracy is 49.48%.

Despite the fact that we do not know whether rectangular traffic levels are appropriate or not, all three algorithms were able to perform significantly better than the baseline. The baseline only yields an accuracy of 49.48%, whereas the machine learning algorithms have an accuracy around 80%. The best performance in this particular case gave the decision tree algorithm J48, but is closely followed by the support vector machine.

The allocation into the single traffic levels is not equally distributed for the rectangular approach, as can be seen in Table 6.1, but this has not been the aim when creating the calculation algorithm. The high amount of traffic level 3 examples is something that has to be checked in a future work, but is likely due to the fact that many of the measurements were taken at around 12pm. At this time the roads are often congested and the rectangular traffic level calculation seems to reflect that.

Accuracy	82.98%
Average Precision	82.72%
Average Recall	82.98%
Average F-Measure	82.80%

(a) J48

Accuracy	82.55%
Average Precision	81.60%
Average Recall	82.55%
Average F-Measure	81.92%

(b) JRip

Accuracy	82.98%
Average Precision	81.27%
Average Recall	82.98%
Average F-Measure	81.41%

(c) LibSVM

Figure 6.2: Triangular traffic levels with 1 minute time interval using 1146 instances. The baseline accuracy is 59.25%.

In the triangular approach the performance of the three algorithms is very similar. Figure 6.2 shows the results for a 1 minute time interval and the triangular traffic level calculation. Interestingly the baseline performs around 10% better than in the rectangular approach. This is due to the fact that the triangular approach has a very high amount of traffic level 3 instances, as can be seen in Table 6.1. As mentioned in the previous section, we assume that this reflects high amount of measurements that were taken around midday, when the roads are usually congested.

All three algorithms yield an accuracy between 82% and 83%. J48 and the support vector machine even yield the same accuracy. However, the precision of the J48 algorithm is slightly better.

6.3 Elliptic Traffic Levels

Accuracy	85.25%
Average Precision	84.92%
Average Recall	85.25%
Average F-Measure	85.03%

(a) J48

Accuracy	83.68%
Average Precision	84.12%
Average Recall	83.68%
Average F-Measure	83.48%

(b) JRip

Accuracy	83.94%
Average Precision	83.34%
Average Recall	83.94%
Average F-Measure	83.28%

(c) LibSVM

Figure 6.3: Elliptic traffic levels with 1 minute time interval using 1146 instances. The baseline accuracy is 41.97%.

Figure 6.3 shows the evaluation of the elliptic approach with 1 minute interval. The elliptic approach is the one that we found closest to the actual conditions on the roads. All measurements that were used to create this evaluation were taken either around 10am - 12am or 1pm - 3pm. The traffic at these times has always been between mediocre or highly congested. We were never able to capture a real traffic jam with our measurements. Further research has shown, that the most traffic can be measured early in the morning around 5am - 6am. This is why there are very few instances that fall into traffic level 4. Traffic level 1 also had very few instances. This was fixed by measuring a road in the time from 12pm to 3am, as there is very low traffic in this time. This could not been done for traffic level 4, because the time for this work was to short.

Within this approach we were able to reach the highest accuracy out of the three different traffic levels calculations, but this does not necessarily mean that this approach is the best. We can not rank the different approaches by the performance of the machine learning algorithms. We plan to research this question in the near future by manually observing traffic on a road and subjectively deciding what traffic level is appropriate for that situation. We can then compare this observations to the three approaches from this work and decide which are appropriate and which are not.

The performance of the three algorithm was again very similar. Once again the decision tree algorithm J48 had the edge over the other two. JRip and the support vector machine perform almost equally.

Because we found the elliptic approach most suitable we also evaluated it for bigger time intervals such as 5 and 15 minutes, which will be explained in the next two sections.

6.4 Elliptic Traffic Levels with 5 Minute Time Interval

Accuracy	78.65%
Average Precision	78.37%
Average Recall	78.65%
Average F-Measure	78.49%

(a) J48

Accuracy	75.80%
Average Precision	76.24%
Average Recall	75.80%
Average F-Measure	75.70%

(b) JRip

Accuracy	83.63%
Average Precision	82.86%
Average Recall	83.63%
Average F-Measure	83.15%

(c) LibSVM

Figure 6.4: Elliptic traffic levels with 5 minute time interval using 281 instances. The baseline accuracy is 43.77%.

We decided to test the elliptic approach on even bigger time intervals. Increasing the time interval significantly decreases the amount of instances in the training set. While J48 was slightly better at 1 minute intervals the support vector machine performs much better than the other two in this scenario. Even with a smaller training set it is able to yield an accuracy of 83.64%, which is almost as high as with 1 minute time intervals.

6.5 Elliptic Traffic Levels with 15 Minute Time Interval

Accuracy	77.97%
Average Precision	77.73%
Average Recall	77.97%
Average F-Measure	77.68%

(a) J48

Accuracy	74.58%
Average Precision	75.80%
Average Recall	74.58%
Average F-Measure	73.76%

(b) JRip

Accuracy	82.20%
Average Precision	81.69%
Average Recall	82.20%
Average F-Measure	81.81%

(c) LibSVM

Figure 6.5: Elliptic traffic levels with 15 minute time interval using 118 instances. The baseline accuracy is 44.07%.

Even when using a 15 minute time interval the support vector machine still performs very well. While the smaller amount of training data has a higher impact on the other two algorithms the support vector machine still yields an accuracy over 82%.

6.6 Traffic Level Distribution

The traffic level distribution was something that was very interesting during research. As earlier mentioned we recognized a high amount of traffic level 3 examples in the first two approaches. This might indicate, that these approaches are inappropriate for the calculation of a representative traffic level. The measurements that were used for the calculation were all taken around midday, which is why we expect a lot of level 2 and 3 examples in the training set.

To get traffic level 1 examples we did an extra series of measurements at night, around 12pm to 3am. All of the three approaches seem to represent this as the amount of examples in traffic level 1 is almost the same for all (Table 6.1).

We never witnessed a real traffic jam during the measurements, which is why we expected a low amount of traffic level 4 examples. The elliptic approach behaved closest to our expectations, which is why we think that this approach is the best out of the three presented ones.

Traffic Level	Rectangular	Triangular	Elliptic (1 min.)	Elliptic (5 min.)	Elliptic (15 min.)
1	241	244	241	49	18
2	57	176	409	106	47
3	567	679	481	123	52
4	281	47	15	3	1
Sum	1146	1146	1146	281	118

Table 6.1: The distribution of traffic levels for the different approaches.

Because of the missing traffic jam, or rather traffic level 4 examples we decided to not include the threshold approach into the evaluation. The threshold approach is only reasonable when we want to detect traffic jams, as measurements which exceed the threshold are automatically assigned to the highest traffic level (traffic jam).

7 Conclusion and Future Work

This work has shown that classifying traffic level with the help of location-aware sensors is possible. The results that the models produced are significantly better than the baseline solution to this problem. Furthermore, it has shown that less complex algorithms such as decision trees and rule based algorithms can yield satisfying results. This is beneficial because they require less time to be calculated.

We had no sources on the calculation of traffic level from such variables as cpl and ups, as there is no publicly available state of the art algorithm. The approaches presented in this thesis are simplistic solutions to this problem. Regardless of the traffic level calculation the machine learning algorithms have shown to be effective in classifying traffic volume with smartphone noise data. There is still room for optimization of the algorithms and the results presented in this thesis are not the top of the line.

This thesis has also taken the results from [1] and evaluated an optimized version of the system on a bigger training set. The reached accuracy of over 80% can be seen as a success in this field. Smartphones can be used as a cheap alternative for inductive loops. They also have the advantage of being commonly available, whereas inductive loops are not installed in every road. Of course, smartphones are not necessary for this approach. The same result can be achieved using simple sound level meters, which can be easily installed at every road.

This method of determining a traffic level might not be as precise as using inductive loops, but it can be useful for such services as "Google Maps" as it can produce real time traffic levels without the use of permanently installed devices such as inductive loops.

The results presented in this work are not complete, as the machine learning model has a lot of room for optimization, but they are sufficient to show, that a machine learning model can be capable of classifying traffic volume based on sound pressure measurements.

There are many ideas that this thesis could not include. We would like to prove the traffic level calculation from Section 5 by watching the traffic on a road for a longer time and and then comparing the data from the inductive loops to the subjective impression that we had watching the road. Our approaches can be vastly optimized when we have a deeper understanding of the inductive loop data. This can be done by observing a road early in the morning when the amount of cars is at its peak. This example of traffic jam and the resulting inductive loop data can then be used to further optimize the traffic level calculation.

The traffic data provided by the inductive loops has made a recent change, as the time interval changed from 15 minutes to just 1 minute. This means the inductive loop data is now saved once every minute. With such low time intervals we do not have to manually divide the training examples into 1 minute intervals. Furthermore, we can calculate a specific traffic level for every one minute interval and do not have to use one traffic level for every 1 minute interval inside of the 15 minutes. This could improve the machine learning as the traffic levels are precisely calculated for the actual 1 minute interval. This is something we want to look at in the near future.

Furthermore, we would like to use an even bigger set of sound pressure measurements. With the increasing amount of users of the da_sense platform this can be achieved in the near future.

The threshold approach could not be evaluated within this work, due to the missing examples for traffic jam. This is also something we want to look into in the future.

Be advised that the results presented in this thesis are dependent on the way that the traffic level is calculated. Whether or not the approaches shown in this work are appropriate for representing traffic level remains to be proven. This was not possible within the time that this work was given.

All in all this work has shown that using smartphones for traffic level classification can be a viable option. It also brings up interesting questions that can be worked out in future projects.

Glossary

Inductive Loop

Inductive loops are used to recognize metal that passes over them. They are mainly used to count the amount of cars that drive on a road.

J48

J48 is an implementation of a decision tree based machine learning algorithm which is part of the Weka framework.

JRip

JRip is an implementation of a rule based machine learning algorithm which is part of the Weka framework.

LibSVM

LibSVM is an open library that contains an implementation of a support vector machine. This library is not part of the Weka Framework and has to be imported manually.

Open Street Map

Open Street Map is a free project in which people around the globe created a map of the world by using collaboratively collected geo data. Everyone can contribute to this project by sending their geo data to the OSM servers in order to make the maps better.

PostGIS

PostGIS is an open source project that adds geographic objects to the PostgreSQL database.

PostgreSQL

PostgreSQL is an open source database that is available for a variety of operating systems. It is based on the SQL syntax.

Acronyms

ARFF	Attribute-Relation File Format
CSV	Comma Separated Values
GIS	Geographic Information System
GPS	Global Positioning System
GUI	Graphical User Interface
ML	Machine Learning
OSM	Open Street Map
SQL	Structured Query Language
SVM	Support Vector Machine
XML	Extensible Markup Language

List of Figures

List of Tables

References

[1] D. Goshev, "Road Traffic Monitoring with Location-aware Sound Sensors", 2012.

[2] R. Sen, B. Raman, and P. Sharma, "Horn-ok-please", *... of the 8th international conference on ...*, 2010. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1814449`.

[3] R. Sen, P. Siriah, and B. Raman, "RoadSoundSense: Acoustic sensing based road congestion monitoring in developing regions", *2011 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks*, pp. 125–133, Jun. 2011. DOI: `10.1109/SAHCN.2011.5984883`. [Online]. Available: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5984883`.

[4] P. Mohan, V. Padmanabhan, and R. Ramjee, "Nericell: rich monitoring of road and traffic conditions using mobile smartphones", *... of the 6th ACM conference on ...*, 2008. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1460444`.

[5] P. Flach, *Machine Learning: The Art and Science of Algorithms that Make Sense of Data*. Cambridge University Press, 2012, ISBN: 9781107422223. [Online]. Available: `http://books.google.co.uk/books?id=VJdQLwEACAAJ`.

[6] D. Powers, "Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation", *Journal of Machine Learning Technologies*, vol. 2, no. 1, pp. 37–63, 2011. [Online]. Available: `http://www.bioinfo.in/uploadfiles/13031311552%5C_1%5C_1%5C_JMLT.pdf`.

[7] University of Waikato, *Weka: Data Mining Software in Java*, 2013. [Online]. Available: `http://www.cs.waikato.ac.nz/ml/weka/` (visited on 07/03/2013).

[8] Open Street Map, *Open Street Map Statistics Report*, 2013. [Online]. Available: `http://www.openstreetmap.org/stats/data%5C_stats.html` (visited on 05/06/2013).

[9] Telecooporation Group TU Darmstadt, *da_sense Project*, 2013. [Online]. Available: `http://www.da-sense.de` (visited on 05/05/2013).

[10] M. Wlotzka, "Delay-tolerant Data Transmission in Mobile Sensor Networks", 2013.

[11] I. Schweizer, R. Bärtl, and A. Schulz, "NoiseMap-Real-time Participatory Noise Maps", *Proc. 2nd Int'l ...*, 2011.

[12] J. Gajda, R. Sroka, and M. Stencel, "A vehicle classification based on inductive loop detectors", *..., 2001. IMTC 2001. ...*, pp. 460–464, 2001. [Online]. Available: `http://ieeexplore.ieee.org/xpls/abs%5C_all.jsp?arnumber=928860`.

[13] Telecooporation Group TU Darmstadt, *Darmstadt traffic raw data*. [Online]. Available: `http://www.da-sense.de/trafficdata/` (visited on 06/29/2013).

[14] R. O. Obe and L. S. Hsu, *PostGIS in action*, ser. In Action. Manning Publications Company, 2011, ISBN: 9781935182269. [Online]. Available: `http://books.google.de/books?id=4kEBRQAACAAJ`.

[15] I. N. Bronštejn, *Taschenbuch der Mathematik*. Deutsch, 2008, ISBN: 9783817120079. [Online]. Available: `http://books.google.de/books?id=VnsL9p8hXfQC`.

[16] W. H. Press, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007, ISBN: 9780521880688. [Online]. Available: `http://books.google.de/books?id=1aAOdzK3FegC`.

[17] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines", *ACM Trans. Intell. Syst. Technol.*, vol. 2, no. 3, 27:1–27:27, 2011, ISSN: 2157-6904. [Online]. Available: `http://doi.acm.org/10.1145/1961189.1961199`.