Creating noise pollution maps based on user-generated noise measurements

Erstellung von Lärmkarten basierend auf nutzergenerierten Lautstärkemessungen
Bachelor-Thesis von Jakob Karolus
Mai 2013

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Telecoorperation Group

Creating noise pollution maps based on user-generated noise measurements
Erstellung von Lärmkarten basierend auf nutzergenerierten Lautstärkemessungen

Vorgelegte Bachelor-Thesis von Jakob Karolus

1. Gutachten: Prof. Dr. Max Mühlhäuser
2. Gutachten: Dipl.-Wirtsch.-Inform. Axel Schulz

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 15. Mai 2013

____

(J. Karolus)

Contents

1 Introduction **4**

2 Basics **5**

2.1 Participatory sensing 5

2.2 Machine Learning and Data Mining 5

2.3 RDF and SPARQL 7

3 Related work **9**

3.1 Participatory sensing and noise mapping 9

3.2 LinkedGeoData and its application scenarios 13

3.3 Data quality of OpenStreetMap and usage scenarios 14

4 Architecture **16**

4.1 Data sources 17

4.1.1 NoiseMap 17

4.1.2 OpenStreetMap 18

4.1.3 LinkedGeoData 19

4.1.4 “Deutscher Wetterdienst” 20

4.2 Implementation 21

4.2.1 Query and data processing pipeline 21

4.2.2 Querying of data sources 21

4.2.3 AbstractAttribute - How to model features 27

4.2.4 DataBuilders - Generating a custom arff file 30

5 Evaluation **32**

5.1 Feature selection and optimization 32

5.1.1 Single features 32

5.1.2 Feature compositions 38

5.1.3 Different sound level distributions 40

5.1.4 Different classifiers 42

5.2 Evaluation summary 43

6 Applications **45**

7 Conclusion and future work **47**

Abstract

Environmental pollution has become a rising concern during the last decades. Especially in big cities with their huge traffic volume, the pollution by the generated noise has become an imminent health threat to the citizen [40].

To protect their citizens, states in the EU have agreed on a directive which forces them to curtail the threat that is noise pollution [31]. First and foremost, this gives citizens the right to be informed about noise threats as every state has to provide noise maps for area of high population density. However, there are three major problems with this approach. Primarily, those maps only provide sparse coverage since rural areas are not included and they also exhibit a relatively long update cycle. Last but not least, recording the maps involves expensive high quality sensors and human resources, resulting in a not negligible financial burden.

In this thesis, we introduce an approach making use of machine learning techniques and collected noise measurements by a participatory sensing application to determine the noisiness factor of an area. By using the collected sound data in conjunction with further information on the vicinity such as nearby streets or buildings, we were able to create a machine learning model able to predict a sound level with 80.9% accuracy. As this approach is very cost-efficient it can easily be used as an addition to common techniques for recording noise maps as well as a standalone application.

Zusammenfassung

Das Thema Umweltverschmutzung ist in den letzten Jahren zu einem ernstzunehmenden Problem herangewachsen. Vor allem in Städten mit großem Verkehrsaufkommen ist die Belästigung durch den hervorgerufen Lärm zu einer Bedrohung der Gesundheit der Einwohner geworden [40].

Zum Schutze ihrer Bevölkerung haben die Mitgliedsstaaten der EU eine Richtlinie verabschiedet, welche die zunehmende Lärmbelästigung eindämmen soll [31]. Durch Lärmkarten von Ballungszentren sollen Bürger über Lärmquellen informiert werden. Dieser Ansatz leidet jedoch unter drei grundsätzlichen Schwächen. Zum einen gibt es für ländliche Gebiete keine Lärmkarten und des Weiteren verhindert der meist lange Aktualisierungszeitraum eine genaue Bestimmung der Stärke der Lärmbelästigung. Zudem werden teure Qualitätssensoren sowie Fachkräfte zur Aufnahme einer Lärmkarte benötigt, was eine nicht vernachlässigbare finanzielle Belastung darstellt.

In dieser Arbeit stellen wir einen Ansatz vor, welcher mithilfe gesammelter Lautstärkemessungen durch eine "Participatory Sensing" Anwendung den Lärmfaktor einer Gegend bestimmen kann. Hierzu verwenden wir Techniken des Maschinellen Lernens und reichern unsere Messungen mit einer genauen Beschreibung der Umgebung des Messorts an, beispielsweise indem wir Straßen und Gebäudetypen in der Nähe ergänzen. Dadurch konnten wir ein Modell erstellen, welches den Lärmpegel mit einer Genauigkeit von 80.9% vorhersagen kann. Da unser Vorgehen sehr kostengünstig ist, kann es leicht als Ergänzung zu herkömmlichen Methoden zur Erstellung von Lärmkarten oder als eigenständige Applikation eingesetzt werden.

1 Introduction

Motivation

Environmental pollution has been one of the major topics over the last decades. As the earth's population increases rapidly, protecting the environment for further generations is vital. Besides air pollution and the resulting climatic change, noise pollution has been a rising issue in recent years. High levels of noise exposure can influence productivity and social behavior in a negative way, apart from degrading the ability to hear [40]. For this reason the European Community passed the directive 2002/49/EC [31], which declares noise protection as one necessary objective to achieve a high level of health and environmental conservation.

The directive imposes several actions to be made upon member states, including the mapping of noise in larger cities via noise maps. On the basis of these maps, the countries can formulate plans to counter the threat that is noise pollution [31]. In Germany, strict threshold values were introduced concerning the construction of streets and railways [28]. Exceeding these limitations would require additional measurements such as anti-noise barriers alongside the road to protect citizens.

Although the prospect of noise maps to curtail pollution seems promising, there exists one major problem concerning this approach. To generate accurate and up-to-date maps, a huge network of noise sensors is needed. As this poses a financial burden onto local authorities or authorized companies, only a limited number of sensors is used while using simulations to interpolate missing measurements [29]. Furthermore an overlong update cycle prevents accurate noise maps when needed [31].

In this thesis, we present an approach to create noise maps based on sound measurements recorded by a participatory sensing application. Using machine learning techniques, we establish models to predict sound levels and generate noise maps based on those predictions. Additionally, we include the vicinity of a measurement location as a knowledge factor by identifying possible noise sources and including them into our model, leading to a more accurate prediction. On the one hand, our approach could be used as a standalone application, e.g. providing noise levels for houses and apartments as an additional advertisement factor or for simulating noise changes before building a new office tower. On the other hand, the generated maps could be utilized in conjunction with already present noise maps delivering correlation data for areas, which were not measured by costly noise sensors but rather calculated with models [29]. On the basis of these results the original measurement plans could be adapted to achieve a better coverage of the relevant area by using even fewer sensors and thus decreasing the financial burden.

Approach

To reach the goal of a precise model predicting noise levels, we make use of an already existing participatory sensing system called *NoiseMap*[1], supplying us with sound measurements tagged with timestamp and GPS location. Using this dataset as training background for our model, we aim to predict noise levels for arbitrary locations and time. However, we expect that this initial model will lack in accuracy as the amount of information is simply not sufficient enough. For this reason, we analyzed several possible noise sources in the vicinity of a location, which could directly or indirectly influence the recorded sound data and could be used to enrich the dataset generated by *NoiseMap*. In doing so, we were able to establish a model predicting noise levels with much higher accuracy.

Based on our analysis, we decided to examine nearby building types using *LinkedGeoData*[2], weather data provided by *"Deutscher Wetterdienst"*[3], as well as nearby streets using *OpenStreetMap*[4] with regard to their usefulness in this scenario.

As we do not use any domain knowledge of sound propagation in our approach, we feature a thorough evaluation of various models. Each one making different usage of our data sources and interpreting them in a different way. To be able to evaluate arbitrary compositions of noise sources in fast succession, we developed a prototype called *LOCAL*[5], which administers our data sources and is able to preprocess the data, making it suitable for machine learning purposes.

Structure of this work

As an introduction to the topics of machine learning and participatory sensing themselves, Chapter 2 and 3 will provide insight into needed basics as well as related work in this area, following up with an overview over the architecture (see Chapter 4). While Section 4.1 will present all of our data sources, the next section (4.2) will focus on implementation aspects of our prototype. In Chapter 5 we will closely examine the usefulness of our approach and evaluate refinements. We will also have a quick glance at possible visualization aspects, before we conclude in Chapter 7 and follow up with a look at future work.

[1] http://www.tk.informatik.tu-darmstadt.de/de/research/smart-urban-networks/noisemap/ [accessed on 15.05.13]
[2] http://linkedgeodata.org/ [accessed on 15.05.13]
[3] http://dwd.de/ [accessed on 15.05.13]
[4] http://www.openstreetmap.org/ [accessed on 15.05.13]
[5] LOCation Aware Loudness

2 Basics

First, we would like to introduce some necessary basics involved with our approach. Since participatory sensing plays a vital part for collecting our initial set of noise data, the first section provides some insight into the concept itself and its recent development. As mentioned in the introduction, we utilize machine learning techniques to extract a connection from the vicinity of a location to its noise level. Therefore we will introduce the concept of machine learning and popular algorithms. The last section covers an explanation of RDF and SPARQL, as we encounter both when querying data sources in the “Semantic web” [7].

2.1 Participatory sensing

The rise of smartphones over the last decade coupled with their huge sensing capabilities has opened up a new perception on sensor networks. Given the right architecture, mobile devices are able to replace expensive sensor equipment, while still providing a huge network of sensing nodes. Beside the integrated sensor possibilities, smartphones offer high computation power and connectivity. This speeds up data processing and strengthen the aspect of “shareability” [5]. Being able to stay “on the grid” and share collected data ultimately provides a stable and consistent sensing network. [5]

Since participatory sensing focuses on interactive networks – directly involving the user – there needs to be some form of incentive. Especially when addressing a large community where the motivations for participating differ substantially [18]. Often civic concerns such as environmental issues are used as a motivational factor [5]. In big cities, air and noise pollution are omnipresent and thus citizens become more aware of this threat. On the other side, in areas of no imminent environmental risks, other incentives must be found in order to get the public to participate. A popular idea is to introduce a competition between data contributors, rewarding them for collecting measurements. [26]

Apart from data quantity, its quality is a vital factor, too. Here we see the shortcomings of an approach interactively involving users of mobile devices. It is far more difficult to achieve reliable measurements when one cannot directly control the nodes of the sensor network. As a result, the input data may be corrupt or simply to sparse in certain areas, which would require additional steps to ascertain a high data quality. [5]

In this work, we present several participatory sensing systems making use of mobile devices to gather data in Chapter 3 and highlight similarities as well as differences to our approach.

2.2 Machine Learning and Data Mining

In a world of data, finding patterns or concepts explaining a given set of data can certainly pose a huge challenge. Especially since estimations show that the amount of information stored in databases worldwide doubles approximately every 20 months [15]. However, having access to more data does not necessarily increase our knowledge. We are basically “drowning in data yet starving for knowledge”[6]. To counter this, we need to find patterns within the dataset to extract usable information, which allows us to establish a concept and makes data “explainable”. In our scenario, we could then explain why a specific location is e.g. especially noisy.

Discovering those patterns is defined as “Data Mining”, describing an at least semi automatic process to scan data for meaningful patterns [44]. As there exist many different approaches to “mine” data, the structural representation of patterns differs naturally. Common ones are represented through a list of rules or a tree structure capturing the concept. “Machine Learning” is about constructing these concept automatically and applying them to unseen data. However, as there are infinitely many data mining problems, there cannot be one universal learning scheme. Since every algorithm has its own specific bias, selecting the appropriate one is the user’s duty, making data mining an experimental science. Yet there are many tools available – e.g *Weka*[7] – which can assist the user in this matter.

Learning algorithms

Developed by the University of Waikato in New Zealand, *Weka*[8] is a workbench providing a variety of implementations for different machine learning schemes suitable for data mining tasks [20]. Based on distinctive features – the attributes – of a dataset, these algorithms are able to predict the outcome of one specific attribute, typically called the “class” attribute. Hence, the algorithm is often referred to as “classifier”. As a complete overview is not the focus of this work, we will only introduce selected algorithms.

J48: J48 is an elaborate decision tree learner originating from C4.5 [32] by Ross Quinlan and is able to cope with “real-world” problems such as numeric attributes and missing values. We favored J48 due to a fast creation phase compared to other classifiers, which enabled us to perform an in-depth evaluation of our results.

In general, there are two steps when it comes to Decision Tree Classification. At first it is necessary to construct the decision tree using a recursive divide-and-conquer algorithm. Starting with the initial set of data, the classifier tries to

[6] anonymous

[7] http://www.cs.waikato.ac.nz/ml/weka/ [accessed on 15.05.13]

[8] Waikato Environment for Knowledge Analysis

find an optimal splitting point (a node in the tree) to divide the set into two parts. Based on heuristics, like Entropy and Information Gain, the classifier ensures a certain purity[9] of the resulting subset with regard towards the class value. For example, when working with a class attribute "play golf" that possesses two possible values `yes` and `no`, finding a splitting point, where one subset contains mostly instances with the class value `yes`, whereas the other subset covers mostly `no` instances, is beneficial. The distribution into a subset is based upon the instances attribute values. Coming back to our example, using the attribute "rainy" as splitting point, we would expect a subset of mostly `no` instances if it is rainy (attribute's value is `true`). [44]

After splitting the dataset, the same step is repeated for each subset individually until either a set is pure or a certain heuristic threshold is reached. Eventually, the generated tree can be used to classify new instances by traversing the tree based on the attribute tests in its nodes. The class value is then calculated based on the composition of instances in the leaf. [44]

JRip and libSVM: Some other classifiers include rule learners (JRip) and support vector machines (libSVM). As we merely utilizes them as a comparison to J48, only a brief introduction will be given.

JRip uses a seperate-and-conquer algorithm to find a list of rules explaining a given dataset. Starting from the initial data, it successively refines one rule[10] by adding conditions until a certain heuristic threshold is reached, e.g. until the rule has at least a precision of 80%. After this step, the algorithm removes all instances which were predicted by the rule and starts over. This process can either be stopped prematurely or when all data instances have been correctly covered. Eventually we obtain a list of rules describing patterns in the data set. Note that when classifying new instances the rules must be evaluated starting at the top of the list. The first rule which matches predicts the class value. [44]

LibSVM [12], a wrapper class for a popular SVM library, covers a more mathematical approach towards machine learning. In principle, a support vector machine classifier tries to find a hyperplane in the vector space of all instances, which separates different classes. Although there exist many possible hyperplanes, extracting the one where the distance between two classes is maximal is most beneficial as classification becomes easier. [44]

Ordinal meta classifier: The "meta" classifier implemented by Eibe Frank and Mark Hall [14] makes use of partially ordered class labels to extract further knowledge. By adding an extra class, Frank and Hall were able to establish an order between class labels. The modified dataset could then be learned by any classifier predicting one of the new labels. Converting them back delivers the original class. This flexibility allows the usage of many different classification schemes within the meta classifier.

Input data

Weka uses a standard attribute-relation file format (arff) to represent datasets. It consists of two section: header and data part. The first specifies the name (line 1) and attributes (line 3-5) of the dataset. Possible attribute types are often numeric or nominal as well as strings or dates. The second part contains information about the data itself (line 7-10). Each line represents one instance and each value – for the prior defined attributes – is separated by a comma. Missing values can be marked by a single question mark. Listing 1 shows a small example arff with two nominal and one numeric attribute as well as missing values for some data instances. Note that there is no indication whatsoever to identify the class attribute. Since the arff file only provides a dataset, specifying the attribute to be predicted with the help of the other attributes, is to be done during runtime. This also enables the user to make different predictions with the same dataset. [20]

```
1  @relation weather
2
3  @attribute outlook {sunny, overcast, rainy}
4  @attribute temperature real
5  @attribute play {yes, no}
6
7  @data
8  sunny,85,no
9  sunny,?,no
10 ?,83,yes
```

Listing 1: Modified arff example of *Weka*'s weather dataset.

Output data and performance indicators

Since the internal model structure of each classifiers differs, there is no method to directly compare them. For example, there is no practical way to compare a decision tree (see Fig. 1) with a set of rules (see Fig. 2). Both could predict class values just as well.

[9] a pure set only contains instances of one class value

[10] rule format: `IF condition1 AND condition2 ... THEN class1`

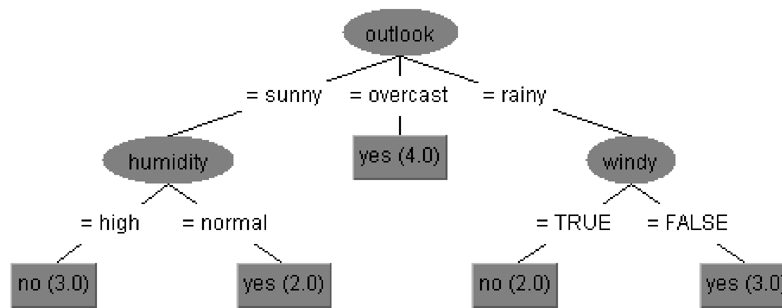


Figure 1: Decision Tree learned by J48, containing internal nodes (ellipses) and leafs (rectangles). The presented number in the leafs conforms to the amount of instances within.

```
JRIP rules:
===========

(humidity = high) and (outlook = sunny) => play=no (3.0/0.0)
(outlook = rainy) and (windy = TRUE) => play=no (2.0/0.0)
 => play=yes (9.0/0.0)

Number of Rules : 3
```

Figure 2: Rule set learned by JRip. The third line contains a default rule which matches any conditions. The numbers correspond to the amount of correctly covered instances (first one) as well as incorrect classifications (second one) for each rule.

Therefore we need another way to determine whether a certain classifier performs better on a given dataset then another one. To address this issue, various indicators can be used to judge the performance of a classifier. At first, a good classifier should make right predictions and thereby the amount of correctly classified examples – the accuracy – is a good indication of how well a classifier is performing. There are many more indexes that can be used to rate the usefulness of a classifier. As most of them correlate between each other, e.g. a high accuracy indicates a low error rate, we refrain from displaying a huge amount of measurements, only mentioning them if we spot irregularities. [44]

However, testing a classifier on the training set itself will probably lead to good results anyway. Hence, we use only a part of the whole dataset to train a prediction model and test it on unseen data, comparing the predicted class value to the real one. A good way to fully utilize all available data for training, is to perform a cross validation on the whole dataset. By doing so, we divide the set into n folds, where we use n-1 fold to train a model and the last fold to test its performance. Note that every fold needs to be the test set at least once, leading to a total of n cycles of creating a model and testing it. By averaging the results, we get a good estimation of the classifiers performance for the desired indicator. [44]

2.3 RDF and SPARQL

RDF[11] – much like XML[12] – can be used to describe web resources and is a vital part of the "Semantic Web". Designed as a standard by W3C[13], RDF is a simple data model to describe arbitrary facts about a resource in the form of triples (a subject-predicate-object expression). Extending XML, it possesses a far more powerful semantic, enabling a more elaborate way to describe entities and to visualize relations between them (often trough graphs). Listing 2 shows a simple RDF document written in XML syntax. [6]

As in any XML document, the first line is occupied by the XML declaration, followed by the root element of the RDF document in line 3 and namespace declarations in line 4 and 5. To describe a resource we use the <rdf:Description rdf:about=resource> ... </rdf:Description> syntax as shown in lines 7-12 and 14-19. Within this block we can specify attributes and their values of the resource, e.g. stating the artist or price of the album. The possible attributes are defined in the respective namespaces, which are not part of RDF itself, but must be designed separately via an ontology,

[11] Resource Description Framework
[12] Extensible Markup Language
[13] World Wide Web Consortium

```
<?xml version="1.0"?>

<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cd="http://www.recshop.fake/cd#">

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Empire Burlesque">
  <cd:artist>Bob Dylan</cd:artist>
  <cd:price>10.90</cd:price>
  <cd:year>1985</cd:year>
</rdf:Description>

<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
  <cd:artist>Bonnie Tyler</cd:artist>
  <cd:price>9.90</cd:price>
  <cd:year>1988</cd:year>
</rdf:Description>
</rdf:RDF>
```

Listing 2: Modified RDF example document from: http://www.w3schools.com/rdf/rdf_example.asp.

e.g. by using RDFS[14], which corresponds to DTD[15] for XML. Note that there exist various methods to actually write down a RDF description. While using a XML-like structure is still readable for humans, listing all possible triples makes things more difficult. [6][41]

However, as RDF is designed to be read by computers anyway, this is were SPARQL[16] comes into play. Utilizing the syntax and semantics specification of RDF, SPARQL enables the user to use SQL-like statements (as shown in Listing 3) to query data sources, which display their data as RDF natively or through middleware. Yet, to fully utilize SPARQL queries, one must be familiar with the underlying RDF ontology just as one must know the database schema to successfully execute SQL queries on a specific dataset. [8]

```
Prefix lgdo: <http://linkedgeodata.org/ontology/>
Select *
From <http://linkedgeodata.org>
{
    ?s a lgdo:Amenity .
    ?s rdfs:label ?l .
    ?s geo:geometry ?g .
    Filter(bif:st_intersects (?g, bif:st_point (12.372966, 51.310228), 0.1)) .
}
```

Listing 3: SPARQL example querying all buildings of type “Amenity” in a given area (line 8).

In our example query (see Listing 3), we extract all buildings of type “Amenity” as well as their name and geometry. Note that in line 8 we additionally restrict the search to buildings around a specific location. As SPARQL needs access to the underlying ontology as well, line 1 contains a namespace declaration to *LinkedGeoData’s* ontology, which we will examine further in Chapter 4.1.3.

[14] Resource Description Framework Schema
[15] Document Type Definition
[16] SPARQL Protocol And RDF Query Language

3 Related work

Our approach covers topics from various research areas. Therefore we structured related work accordingly. First, we will look into work associated with participatory sensing and processing audio data. As auxiliary data sources play a vital part in our approach, we additionally provide some insight into the data quality of two sources, namely *LinkedGeoData* and *OpenStreetMap*, and present applications utilizing them.

3.1 Participatory sensing and noise mapping

Participatory Sensing is a huge and active research area. In our case, we focus on noise pollution mapping applications. The following examples of related work fall into this category.

Laermometer

Laermometer's aim is to provide "noise information for any place in the world" [3]. By collecting noise measurements on-the-go and uploading them to a server, each user can view the noise maps via the web interface. It is also possible to add comments to the measurements, e.g. explaining a particularly high value. However, the authors conclude that the system is yet to be tested in real-world scenarios as it does not address the issue of calibrating different devices or detecting the position of the phone, e.g. whether the phone reside in the user's pocket or hand. Furthermore, a scheme to convert the audio signals to a usable loudness scale (e.g. sound pressure level) is yet to be implemented.

NoiseSPY

A more elaborated work is present by Kanjo in *NoiseSPY* [23], a sound sensing system for monitoring environmental noise. Apart from recording sound measurements by the user, it features a visualization in real-time and a web portal, which allows contributors to access their recordings. Important aspects in Kanjo's work include the personalization of data samples. The application features a personal exposure statistic, allowing a more people-centric approach to noise monitoring. This directly addresses the issue of motivating people to participate, while also providing collaboratively collected measurements via a noise map, shown in Figure 3.

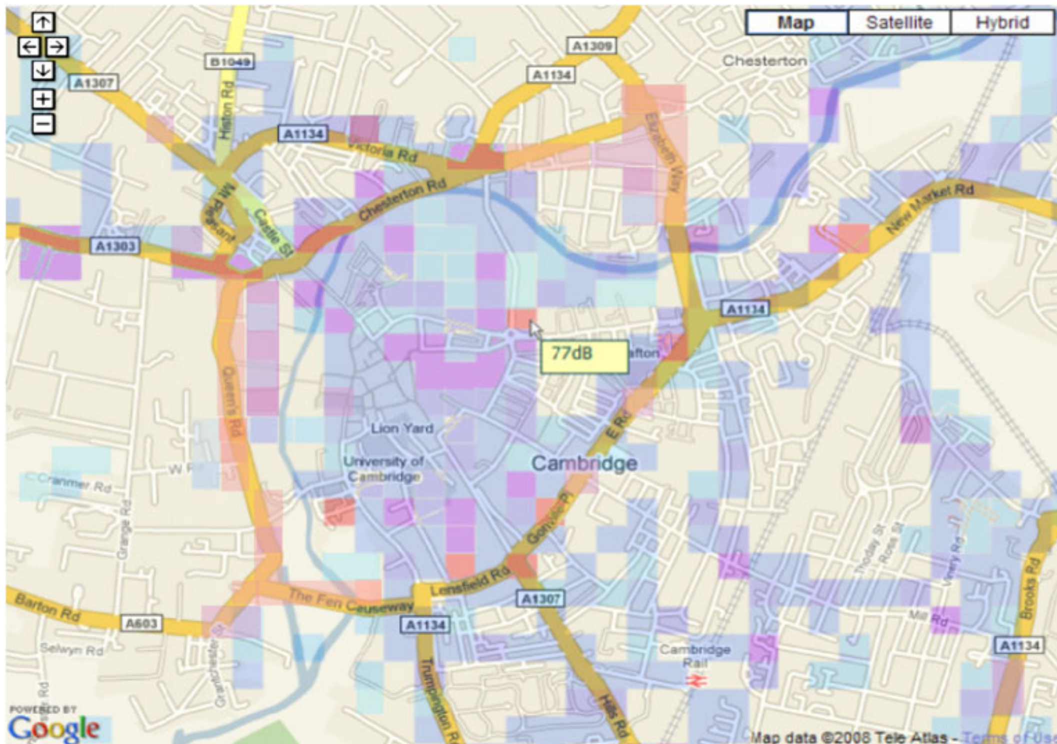


Figure 3: Noise map featured by *NoiseSPY* from a two weeks pilot phase. [23]

To improve data quality, Kanjo uses A-weighting [13] and averages audio signals from the microphone over a given timespan. This conforms to the "long term equivalent noise level" specified in the EU directive [31]. Additionally, before executing a case study, they calibrated the used phones with accurate sound meters. Upon finishing the aggregation of sound samples, the mobile application sends the noise levels to a central server.

The web portal supports two different visualization aspects: a standard noise map and a journey based visualization. The latter provides all data for a specific series of recordings, e.g. a bicycle trip. While privacy is not an issue, if the

user can only view his private recordings, the approach to publicly share measurements needs to protect the privacy of contributors. In *NoiseSPY* a coarse grained privacy policy is implemented by blurring spatial and temporal dimension of noise recordings to hinder user tracking.

A trial run in Cambridge for two weeks with 8 users showed that almost all participants found the idea of monitoring noise levels useful and fun. For the future, the authors want to examine noise interference from user movements or conversation which may lead to compromised measurements, such as a high peak level in an usually quiet area. Applying a correction factor could reduce background noise and help focusing on the real noise source. Additionally, the team wants to address the high power consumption, which degrades usability, and port the application to Android as well as iPhone, making it accessible for a wider range of people. Furthermore, the integrated privacy policy should be extended by informing users actively about the data handling practices as well as using verification protocols to increase data integrity and reduce compromised noise samples. A final aspect addresses the issue of sparsely covered regions by suggesting an interpolation between noise levels and using data aggregation with third party sources, such as air pollution and weather data. In our approach, we pick up this idea by including the vicinity of a measurement's location into noise level calculation as a mean to predict levels for uncovered areas.

NoiseBattle and NoiseQuest

The work of Marti et al. [26] features a prototype to address the issue of motivating users to participate in collecting noise measurements. By applying gamification techniques, the authors aim to get rid of the boredom when doing repetitive tasks such as recording a series of sound levels and thereby motivating users to contribute. As the team does not focus on providing an implementation to capture noise measurements, they use an open source application called *NoiseDroid*[17] to collect data. The important aspects of their work include the realization of gamification techniques in the mobile apps NoiseBattle and NoiseQuest. The former one features a more competitive play style by letting users conquer area and win points. For example, a user can conquer an area by providing more noise samples than any other user in this area. However, other users can still reconquer ares in the same fashion. By additionally providing rewards for certain locations, e.g. allowing the possibility to send "noise" to one's foes, an optimal distribution of measurements can be ensured. Figure 4 illustrates different stages of a NoiseBattle.

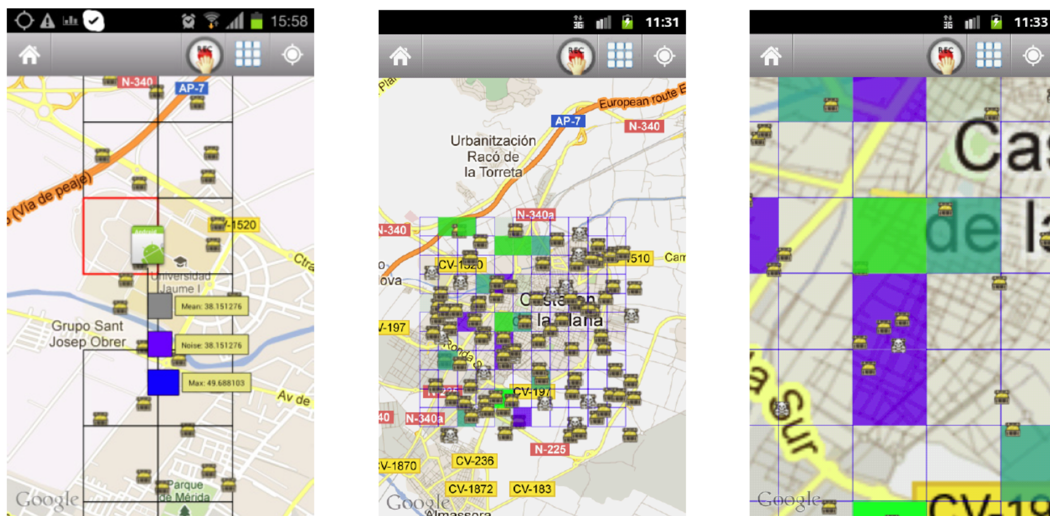


Figure 4: Different stages of a NoiseBattle with three players. Colored areas are conquered by the respective players. [26]

Contrary to NoiseBattle, NoiseQuest pursues a single player oriented approach. In this scenario, the "goodness" of a measurement is more important. By completing quests, challenges and missions the player advances in the game, which follows a story line. As the game progresses, the quests get more difficult and the area increases demanding more commitment by the user to achieve points. At the time of this writing, both game approaches are still in development and not yet publicly available. Possible refinement include the grid mapping onto cities, as this could be adapted to the actual city layout, as well as the idea to decrease the player's points over time, which would encourage a continuously contribution of noise samples. Despite lack of a real-world evaluation, the team concludes that the approach to use gamification as a motivation factor can be applied in other environmental monitoring application as well. An approach also used by *NoiseMap* – the participatory sensing system we used – as the app implements a ranking system rewarding contributors with special titles, like mayor of a city.

[17] https://play.google.com/store/apps/details?id=de.noisedroid&hl=de [accessed on 15.05.13]

NoiseTube

Motivating users to participate in noise sensing, is also a vital part of the work behind *NoiseTube* [24, 25]. Apart from creating noise maps through sharing public measurements, the app aims to provide a personal noise exposure for ever contributor as shown in Figure 5. By doing so, the authors actively involve the users and hope to raise the awareness of people towards noise pollution. For example, tagging noise sources or providing an annoyance rating delivers important semantic information for the underlying system. The noise map itself (see Fig. 6) can be viewed via a web interface, providing semantical exploration, like showing only traffic noise on Monday morning, and an additional API to query the huge dataset for scientific purposes.

Figure 5: Personal Exposure featuring current noise level and a histogram. [24, 25]

Figure 6: Map interface of *NoiseTube* showing different noise level in Google Earth. [24, 25]

By focusing a people-centric approach to collect data, the team wants to establish a new social perspective towards noise pollution. Social translucence is one of the important keywords in this case. The users are able to create a personal exposure log (*Elog*) and share their "noise experience" on web platforms, such as Twitter and Facebook. Following this, the authors expect the formation of small communities based on sharing similar noise problems, e.g. a group of villagers living in the entry lane of an airport.

The architecture of the system consists of a mobile application working with a central server. Using an long term equivalent noise level (secondly timespans) corrected by A-weighing [13], the phone uploads the aggregated levels onto the server. By applying preprocessing steps, like automatic tagging (day or night, street location, etc.) and GPS correction (pulling measurements towards streets), the data gets refined for the purpose of displaying an accurate noise map. In future work, the team wants to focus on the problem of calibrating mobile devices. For the moment the app includes

only a limited number of calibration profiles for end devices. Another issue may be the sustainability of human networks for a longer period of time. Therefore extending the *Elog* with more collaborative features to strengthen such networks is the next step. A small pilot project using *NoiseTube* can be seen in *BrusSense*[18] [10]. Furthermore the app is publicly available[19] for mobile phones running Android, iOS or Java ME with currently over 1600 registered users.

Contrary to our approach, none of the mentioned noise pollution mapping system above interpolates missing values. Thereby, they can only display data collected by the user and fail by design if data coverage is sparse for a particular area. Possible methods to address this issue are illustrated in Kaiser's and Pozdnoukhov's work as well as the application *EarPhone* [34, 33].

Kernel stream oracles

Kaiser et al. [22] feature a scalable algorithmic architecture for modeling data streams from various sensing infrastructures. Using kernel methods [21], the authors leverage machine learning to create predictive models for real-time estimation. Possible data stream include, for example, readings from phone sensors (noise levels) or even twitter message generated by users as shown in Figure 7.

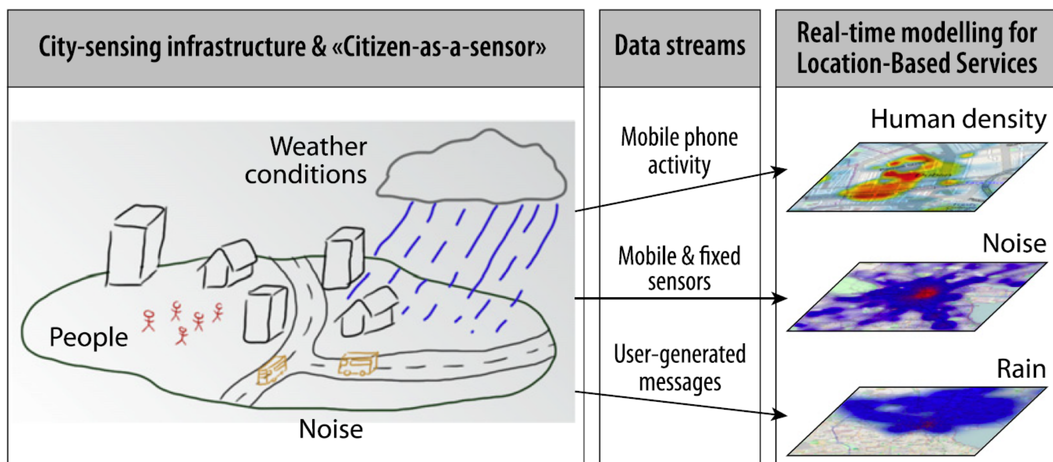


Figure 7: Possible data streams in a city sensing scenario. [22]

To be able to process a large amount of data, a scalable framework using the MapReduce [9] paradigm was used. By distributing the overall calculations on several compute nodes, the team was able to provide a stable and robust system to handle the input of many sensors in parallel. In their paper, the team showcases their approach by establish a real-time noise map based on a continuous stream of sound measurements from mobile devices and stationary sensors. In conjunction with a human density estimation (another showcase presented in the paper), the authors calculate noise levels based on traffic density data provided by a multi-agent [2] simulation of individual vehicles, such as buses. Simulated "ground-truth" data was provided by 5-minute-aggregates of the vehicle count and their estimated noise. The evaluation showed, that the model has difficulties predicting peak noise levels as well as regions which miss recent measurements.

Ear-Phone

Closest to our approach is the urban noise mapping system *Ear-Phone* [34, 33] by Rana et al. The authors utilize a crowdsourcing approach to collect noise measurements. Due to the random distribution of sound samples associated with this kind of approach, they present a reconstruction method for incomplete and missing noise data.

The whole architecture of the system is illustrated in Figure 8. It consists of a signal processing and communication unit on the mobile phone as well as the reconstruction module and visualization on a central server. The signal processing unit is responsible for calculating a long term equivalent noise level. By sampling the signals from the microphone and applying A-weighting [13], the authors ensure the high sound data quality needed for reconstruction. Upon communicating the noise levels to the server, the reconstruction module uses compressive sensing [11] to recover missing samples. Since the team focused on two different method to transmit and recover the sound signals, they feature an evaluation for the so called "projection method", where they use aggregated sound levels as explained above and an evaluation for the "raw-data" method, where no aggregation of sound signals is executed. As a result, the latter has a slight communication overhead, but performs better when it comes to reconstruct missing signals. The method was able to recover noise maps, where 40% of the original data were missing, while only making a negligible error.

[18] http://www.brussense.be/ [accessed on 15.05.13]
[19] http://www.noisetube.net/ [accessed on 15.05.13]

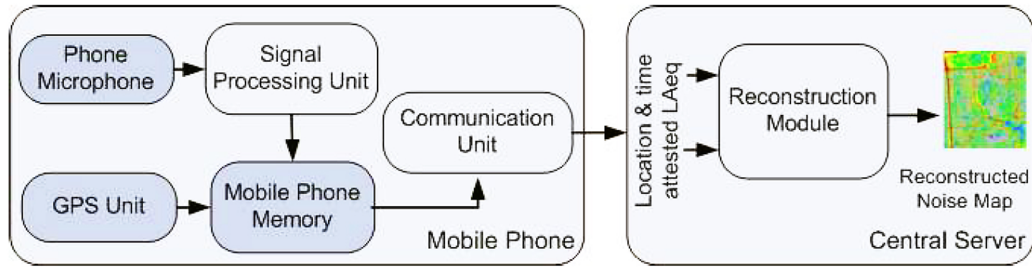


Figure 8: Architecture of *Ear-Phone* showing the separation between mobile phone and server. [33]

Apart from evaluating the performance of their reconstruction accuracy, Rana et al. also consider the possibility of compromised sound levels caused by the placement of the mobile device (e.g. in a backpack or in the user's hand). Surprisingly, the difference was rather slim, with mostly the variance of the measurements increasing. Nevertheless, the team concluded that adding some kind of "context-awareness" would benefit their approach. Other open factors include the optimization of battery and CPU usage, as the high consumption only allowed to take measurements every 30 seconds, and the need for an automatic calibration. Since the authors operated with only two phones (Nokia N95 and HP iPAQ) a simple calibration with a sound level meter was sufficient in this case.

Ear-Phone provides an elaborate method to reconstruct missing audio samples, however, the approach is not designed to predict noise level at completely unknown location in contrary to our system.

Summary of noise mapping systems

Table 1 provides an overview over the presented related work in this section. Additionally, we compare the different systems – including our approach – with regard to selected features they provide. These include a form of noise mapping, supplied by every system, but also more distinctive features, like predicting noise levels for unknown locations or reconstructing missing audio samples (only provided by *Ear-Phone* [34, 33]). As shown in the comparison, our approach is the only one which includes the vicinity of measurements for predicting sound levels and thus enables us to achieve a higher accuracy.

	Noise mapping	Evaluation on real-word data	Reconstructing missing audio samples	Predicting levels through machine learning	Including the vicinity of measurements
[23] NoiseSPY	✓	✓			
[24, 25] NoiseTube	✓	✓			
[22] Kernel stream oracles	✓			✓	
[34, 33] Ear-Phone	✓	✓	✓		
Our approach using NoiseMap	✓	✓		✓	✓

Table 1: Comparison of selected related work and our approach with regard to different features.

3.2 LinkedGeoData and its application scenarios

As mentioned before, the inclusion of information about the vicinity is a vital part of our approach, e.g. using *LinkedGeoData* (LGD) to extract nearby buildings. As we will have a detailed look at LGD itself in Chapter 4.1.3, we introduce some third party systems, which also use LGD, in this section to provide an overview over the capabilities of LGD.

STEVIE[20] [4], is an Android app developed by Braun, Scherp and Staab at the University of Koblenz. It allows the user to create, edit and share semantic points of interest as presented in Figure 9. The ontology for the approach is created collaboratively by using *DBpedia*[21] and LGD on the one hand, but also including classifications by users. To improve the quality of the collected points of interest, data mining techniques for clustering are used. Furthermore the application allows the user to create events and thereby combining temporal and spatial information.

20 http://tiny.cc/stevie10 [accessed on 15.05.13]
21 http://dbpedia.org/About [accessed on 15.05.13]

Figure 9: Creating a point of interest in *STEVIE*.

Another system that uses LGD to enrich a map interface is the augmented reality browser *Layar*[22]. The app uses LGD to display the name of objects captured by the phone camera and to classify them [39]. Two discontinued application include *BeAware*, a website to manage events utilizing LGD's ontology, and *Vicibit*[23] which allowed to create customized views of LGD's dataset on a map, e.g. only displaying restaurants [39].

3.3 Data quality of OpenStreetMap and usage scenarios

Additionally to *LinkedGeoData*, we use *OpenStreetMap* (OSM) to extract information about streets in the vicinity. In 2010, Zielstra and Zipf evaluated the data quality of the project [46]. In their work, the dataset of OSM was tested against the set of a commercial provider (TeleAtlas[24]). Similar to Haklay [19], who investigated OSM data quality against Ordnance Survey datasets, they used the total street length difference as a quality indicator. Furthermore, using a circular buffer method around selected German cities, which calculated the length difference for various radii, enabled them to simulate the transition from urban to rural ares. Their results showed, that OSM contains less data than the commercial application with regard to the total length, but demonstrates tremendous growth over the last years. Especially in rural areas, as the buffer method illustrated, the dataset of OSM was sparse.

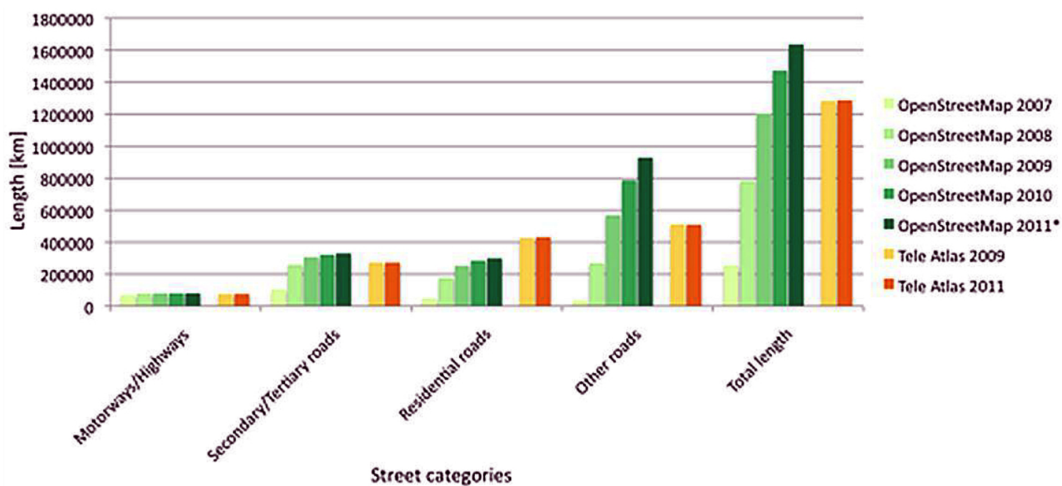


Figure 10: Evolution of OSM street network in comparison to TeleAtlas for different categories.

[22] http://www.layar.com/ [accessed on 15.05.13]
[23] exhibit your vicinity
[24] united with TomTom in 2011

In 2011, the same team with the addition of Neis, illustrated the evolution of OSM with regard to its data quality form 2007 to 2011 [27]. In this newer work, OSM claimed the leading position in total street length (with a difference of 27%) against a commercial system from TomTom, while only taking second play in car-routing-related streets with a 9% difference. Figure 10 shows the development of the OSM street network in comparison with the commercial system TeleAtlas for different categories. According to the team's estimation correlated with the current growth of OSM's dataset, the project could claim the top position in both sections by mid 2012. In big cities, the dataset of OSM related to car navigation is already superior to the commercial system.

An approach using OSM for data mining task is presented in [18], where Hagenauer and Helbich utilize artificial neural networks and genetic algorithms to predict the delineation of urban areas. By using urban land use datasets – provided by GMES[25] – in conjunction with the data from OSM, they were able to achieve an overall squared correlation coefficient R^2 of 0.589. However, R^2 ranged from 0.129 up to 0.789 indicating a spatial heterogeneity in model performance. As previously mentioned, this is due to the sparser dataset of OSM in rural areas.

[25] Global Monitoring for Environment and Security; now Copernicus: http://copernicus.eu/ [accessed on 15.05.13]

4 Architecture

In this thesis, we present an iterative approach to create a model which can predict the noise level of a location by using sound measurements collected by a participatory sensing application. This enables us to generate urban noise maps even for areas with only a few or no measurements at all. As mentioned in the introduction (see Chapter 1), solely relying on sound measurements was not sufficient to construct an accurate model. Therefore we suggest to utilize the vicinity of a measurement's location to find out why a certain place is especially noisy or quiet. Since there exist many possible noise sources, a prior selection was necessary to preserve the feasibility of our method.

Eventually, we extracted three additional sources (left side of Fig. 11), which – in our opinion – greatly influence the sound level of a location. Consider, for example, a noisy motorway nearby or a park during a sunny and warm day, where we would except a rather high noise level compared to colder days. To accommodate future noise sources, we designed the architecture to be highly modular to support any input source as long as they conform to the interface used by our prototype *LOCAL*. The current sources provide information on building and streets in the vicinity as well as the current weather situation. These will be presented in Chapter 4.1.

After identifying our noise sources, we need to convert the obtained data information into a format usable for machine learning purposes. This process is called "Feature Generation" (see Fig. 11). As explained in Chapter 2.2, *Weka* uses a list of attributes and their corresponding values to describe datasets. Following this concept, we need to find a good conversion of the data information provided by our sources into a fixed list of attribute and values. For example, the feature "Weather" may be described as a list of meteorological measurements – such as temperature, sunshine duration, etc. – and their respective values. In Chapter 4.2, we illustrate the process of querying our data sources and extracting the needed information. Additionally, we will present possible strategies on modeling features as well as depict advantages and disadvantages that come along with each strategy.

Since we do not know exactly how the noise level gets influenced by environmental factors, we propose an iterative "Feature Selection and Refinement" (see Fig. 11) in Chapter 5. We evaluate the usefulness of various feature compositions and their resulting models until we ascertain the best possible representation of each feature. After doing so, we can use the optimal model to predict noise levels of unknown locations, for example to create noise maps of various cities as shown in Chapter 6.

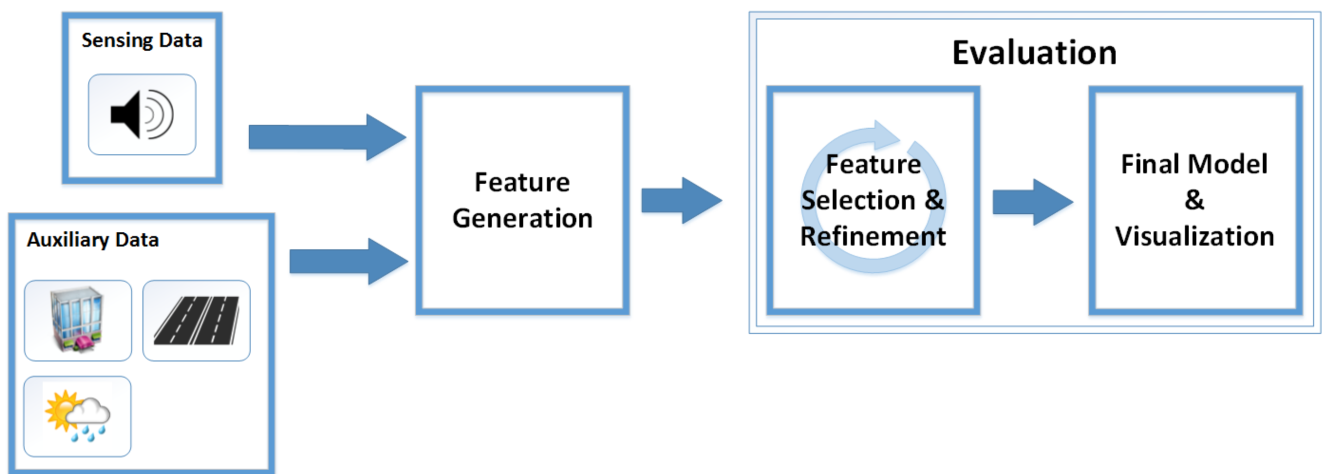


Figure 11: Architecture of our approach showing the steps from querying data information over feature generation, selection and optimization to a final model and its visualization.

4.1 Data sources

As outlined in the introduction, we utilize several data sources beside *NoiseMap* itself to predict sound levels. By identifying possible noise factors and connecting them to appropriate data sources, we are able to enrich our initial dataset with further knowledge and increase the accuracy in predicting noise levels. In this chapter, we introduce our initial data source – *NoiseMap* – as well as all secondary sources we used in our approach. The process of querying these sources will be explained in Chapter 4.2.2, as this is part of the implementation.

4.1.1 NoiseMap

NoiseMap [36, 37, 35] is a participatory sensing system to gather noise data, first released for Android[26]. Users are able to record series of noise data using the microphone of their smartphones (see Fig. 13a). As *NoiseMap* is connected to the *da_sense*[27] platform, every user can review his data via an unique account. This allows them to establish their own “noise profile” by running several recordings. Additionally, a user can share their recordings, which get used to establish a noise mapping as shown in Figure 12. To increase participation, *NoiseMap* also provides a ranking system (see Fig. 13b) and implements gamification techniques (as presented in [26]), rewarding those which are most active in collecting data. The huge amount of noise measurements (75000) during the time period from February until October 2012, confirmed the incentives to motivate citizens to be effective. We use this snapshot as our ground truth data to judge the predicted noise levels at the same locations. Thereby we are able to estimate the performance of our approach, as we can ascertain if a prediction was right or wrong.

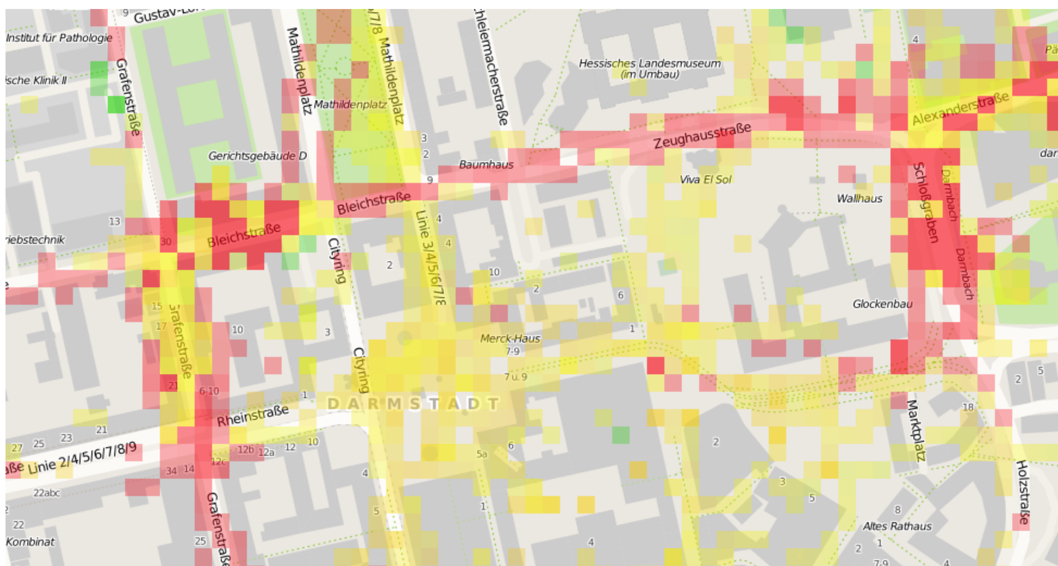


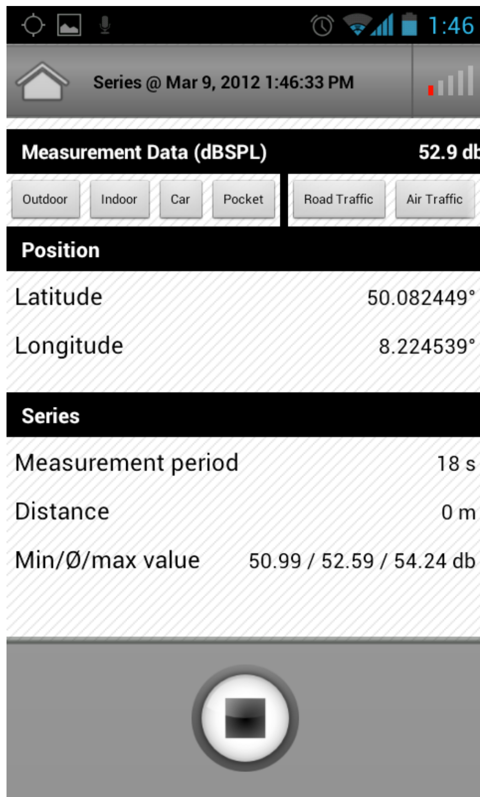
Figure 12: Noise map of the city center of Darmstadt by *da_Sense*.

Compared to other noise recording system presented in Section 3.1, *NoiseMap* focuses on providing accurate noise levels, including a multi-point calibration of devices and frequency adjustments. First, sampling the discrete audio signals over a given timespan provides us with an initial loudness level. This value is measured in dbFS (db full scale). Note that 0 dbFS corresponds to the maximum loudness value perceivable by the phone. All lower values are negative. To calibrate a device, we need to convert the values into the sound pressure level (dbSPL). [36, 37]

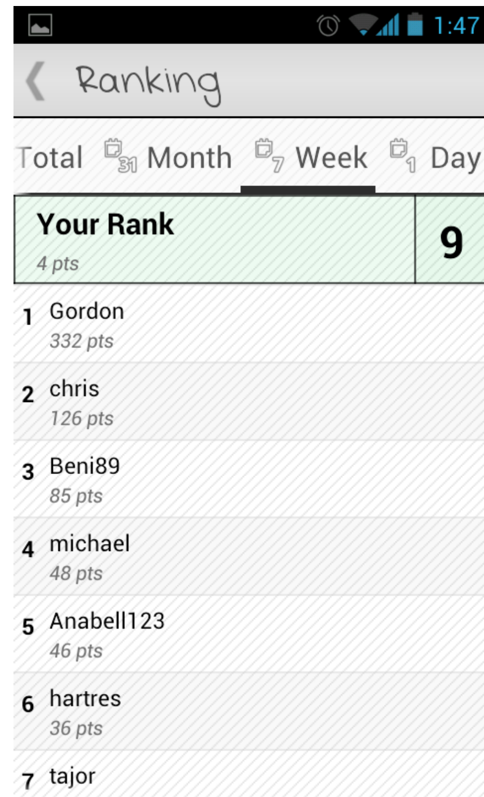
We can distinguish between single-point calibration, e.g used in *Ear-Phone* [34, 33], or multi-point calibration used in *NoiseTube* [24, 25]. The former method includes a single calibration factor for the whole frequency range. As there is no direct relationship between dbFS and dbSPL, this calculation is erroneous. Multi-point calibration tries to minimize this error by using multiple frequency points for calibration purposes, calculating a correction factor for a frequency range. [35]

By additionally capturing the frequency response of the phone and applying a Fast Fourier Transformation (FFT) to convert the input signal from the time to the frequency domain, *NoiseMap* features multi-point calibration while also providing correction of middle frequencies (the range of the human voice), as those are amplified by the phone for better communication. It is now possible to apply a calibration factor for each frequency range separately. Conveniently, applying A-weighting in this domain is also more accurate than in the time domain. The final loudness is then calculated

[26] https://play.google.com/store/apps/details?id=de.tudarmstadt.tk.noisemap [accessed on 15.05.13]
[27] http://www.da-sense.de/ [accessed on 15.05.13]



(a) Recording a series of noise measurements.



(b) Ranking system showing contributors and their points.

Figure 13: Android app *NoiseMap*

from the corrected signal after using an inverse FFT to get back to the time domain. As of now, *NoiseMap* is the only noise monitoring system that provides such a sophisticated calibration method. [35]

4.1.2 OpenStreetMap

Inspired by *Wikipedia*[28], *OpenStreetMap*[29] (OSM) strives to provide a free to use, editable map of the word. Since its foundation in 2004 [46], the project grew to be one of the most promising VGI[30] [17] projects, exceeding the number of one million contributors in January 2013 [42]. OSM is licensed under the ODbL[31] and similar to *Wikipedia's* approach, every OSM user can edit the data and thereby contribute to the whole project.

Keeping the necessary background knowledge to a minimum, OSM introduces a simple yet very powerful data structure to model the world. There are three geographical entities that can be edited by contributors. `Nodes` represent the basis as the most primitive data type, resembling only a geographic location via its coordinates in the WGS84[32] reference system. Combining two or more `nodes` leads to `ways`, which can be used to represent linestring geometries, such as streets. The most complex entities are `relations` which combine an arbitrary amount of `nodes`, `ways` or even `relations` themselves. These can be used to model multipolygons for example. [39]

Although this structure enables us to model a geographically correct map, we are still missing vital properties of the described objects. To address this issue, OSM also features a tagging system. Each tag consists of a key-value pair, which provides domain knowledge about the specified object as illustrates in Figure 14. Contributors can use these tags to further characterize OSM entities (`nodes`, `ways` or `relations`) via properties. Otherwise it would be impossible to distinguish a motorway from a cycleway, as both are represented as `ways`. Note that a user can add any kind of key-value pair. To control tags to a certain extend, a so called Watchlist[33] provides an overview of tags accepted by the community. [16]

[28] http://www.wikipedia.org/ [accessed on 15.05.13]
[29] http://www.openstreetmap.org/ [accessed on 15.05.13]
[30] Volunteered geographic information
[31] Open Data Commons Open Database License
[32] World Geodetic System 1984
[33] http://tagwatch.stoecker.eu [accessed on 15.05.13]

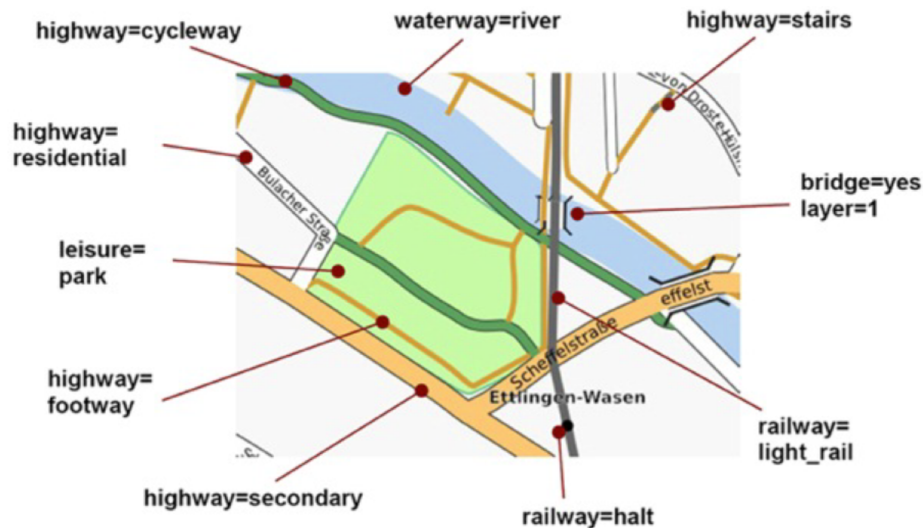


Figure 14: Example of an OSM map with corresponding tags from [16].

As of March 2013, the planet file – the whole database of OSM – contains more than 1.7 billion nodes, 171 million ways and 1.8 million relations [30], which make up approximately 27 GB of disk space. To enable easy access to this vast amount of data, several community websites provide localized data dumps in XML format as well as changesets via the OSM wiki[34], which can be used to keep a local dataset up-to-date. Also provided are tools to parse and process these data dumps, like *Osmosis*[35], enabling everyone to use the available data to its fullest extent.

4.1.3 LinkedGeoData

Initially brought to life in 2009 the *LinkedGeoData*[36] (LGD) project aims to "add[..] a spatial dimension to the Web of data"[1]. Using *OpenStreetMap* to create an ontology and interlinking it with other sources, such as *DBpedia*[37], *GeoNames*[38] and the *Food and Agriculture Organization of the United Nations*[39] has surfaced an "integrated and interlinked geographic dataset for the Semantic Web"[39].

A vital pillar of the project is the mapping of the internal OSM ontology, mainly represented by additional tags of an entity, to a newly created lightweight ontology. Further following the idea of a Sematic Web, LGD provides access to every entity via their respective URI. Each one holds the extracted OSM values mapped to the new ontology, which e.g. links nodes to their building types as well as providing geographical information via coordinates. As long as it is applicable, OSM tags get mapped to RDF properties of the respective entity. [39]

What are the benefits of this modified data structure? Firstly, by providing OSM entities via RDF triples, the system allows us to efficiently search for those relevant to our case. Last but not least, upon finding an interesting entity, we can extract its data and ascertain a semantic context thanks to the ontology. To sum it up, LGD provides us with the means to find interesting entities, particularly buildings, for a given geographical location and exploits their properties in an easy to parse fashion.

By 2011, the LGD dataset contained up to 65 million triples relating to approximately 6.3 million nodes. In the timespan from November 2010 to April 2011, the SPARQL endpoint was queried for a total of 127.000 times showing a drastic increase towards the end of the timespan. This could indicate that the popularity of LGD has further augmented since then.[39]

As far as collecting the necessary data goes, we already mentioned the SPARQL endpoint. However, LGD offers many possibilities, varying e.g. in their granularity and completeness. To get the whole package, one can simple download the complete dataset, which sadly is outdated at the moment[40] and does not provide changesets. Using the live query possibilities such as the REST interface or the SPARQL endpoint seemed more feasible. Due to the powerful query syntax, we chose the latter as our entry point in the world of *LinkedGeoData*. The detailed structure of the implemented adapter and the used queries will be presented in Chapter 4.2.2.

[34] http://wiki.openstreetmap.org/wiki/Planet.osm [accessed on 15.05.13]
[35] http://wiki.openstreetmap.org/wiki/Osmosis [accessed on 15.05.13]
[36] http://linkedgeodata.org/About [accessed on 15.05.13]
[37] http://wiki.dbpedia.org/About [accessed on 15.05.13]
[38] http://www.geonames.org/ [accessed on 15.05.13]
[39] http://www.fao.org/index_en.htm [accessed on 15.05.13]
[40] Dataset is of April 2011

The *LGD Browser*[41] (see Fig. 15) implemented by the *LinkedGeoData* team, features a visualization of known entities on a map and provides the functionality to highlight a shown entity exploiting their properties.

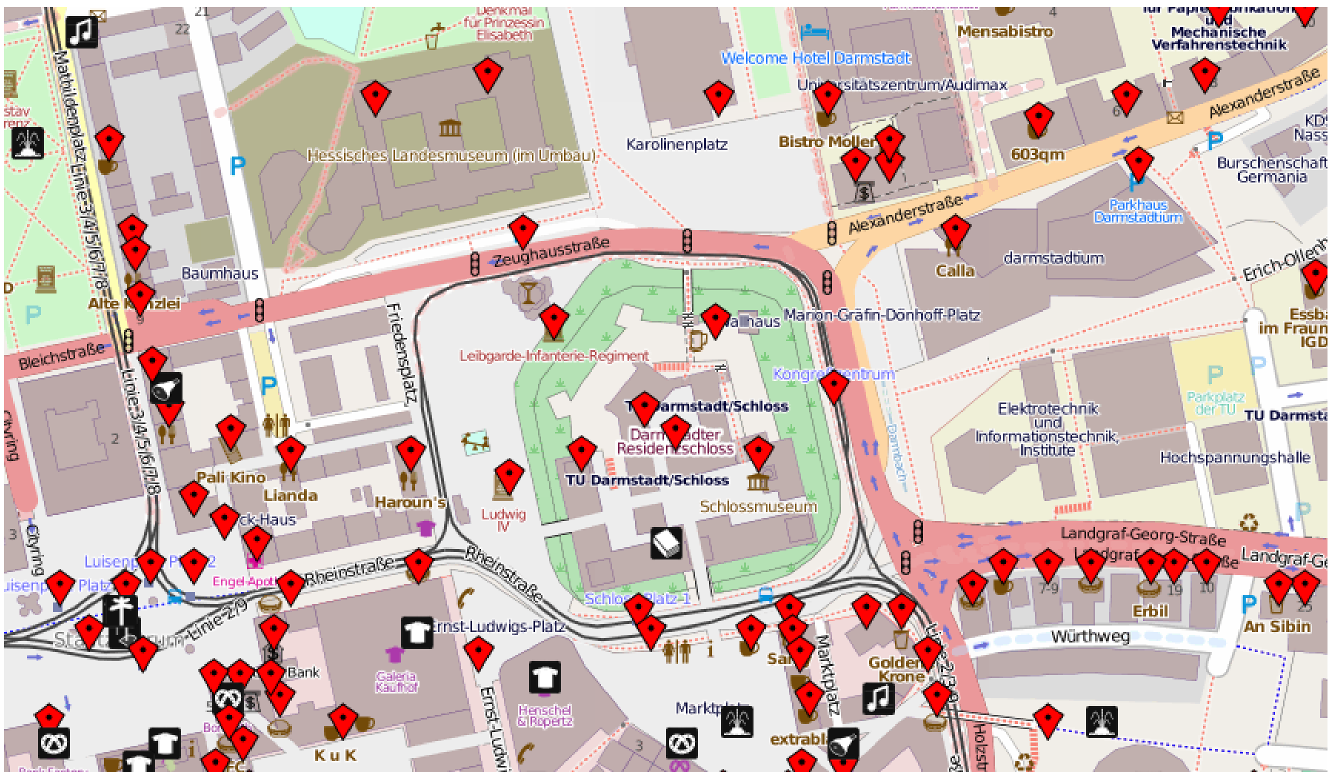


Figure 15: LGD Browser showing the center of Darmstadt. The red rhombuses as well as the black and white symbols mark entities in *LinkedGeoData's* database.

4.1.4 "Deutscher Wetterdienst"

The German meteorological service known as *"Deutscher Wetterdienst"* (DWD) provides us with weather information of a given location. Founded in 1952, the DWD's main duty is to provide meteorological safety for aviation and navy. Furthermore, it issues warnings about upcoming weather events which may endanger the public safety. Most importantly for us in this case is the fact that the DWD also maintains the national climatic archive for Germany. [43]

Since almost all of *NoiseMap's* data is bound to a location in Germany, we considered using a local service providing stable and consistent as well as daily weather information. There are, however, further services providing world-wide coverage such as *OpenWeatherMap*[42], but may lack in density. Aiming at high density coverage in Germany, we decided crawling weather data from DWD.

Although the website of DWD provides historical meteorological information about 78 weather stations, it can be quite a hassle to collect the weather data for a specific location. Since there exists no real API to collect the weather information, e.g. via a REST interface, we developed a crawler downloading the appropriate files from their website. As this is part of the data collection itself, it will be discussed more detailed in Chapter 4.2.2.

41 http://browser.linkedgeodata.org/ [accessed on 15.05.13]

42 http://openweathermap.org/ [accessed on 15.05.13]

4.2 Implementation

In the previous chapter we presented all data sources used in our implementation. This chapter will focus on the techniques and methods used to extract the needed information from those sources as well as introduce the originated features used for the learning process. Since this section focuses on technical aspects of the program, we will also provide some insight into the used class architecture.

4.2.1 Query and data processing pipeline

As we expect the number of our data sources to grow further in the future, we designed our pipeline accordingly. This means that modularity and extensibility is of utmost importance to be able to not only replace current data source, but to accommodate to the needs of future ones. The resulting design strictly separates the process of querying the data sources and establish an internal data structure, from the task of adding refined information for learning into the constructed arff file as shown in listing 4. The internal data structure is provided by the `PointOfInterest` (POI) class, encapsulating one noise datum and exploiting its auxiliary information gathered during the query phase.

```
1  //————————— INIT PHASE ————————//
2  features         //user−selectable
3  sources = extractNeededSourcesFromFeatures(features);
4  List<POI> pois = empty;
5  ArffFile arff;
6
7  //————————— QUERY PHASE ———————//
8  FOR NOISE DATA source
9          FOR every datum
10                 poi = constructPOIFromNoiseDatum(datum);
11                 pois.add(poi);
12
13 FOR ALL REMAINING sources
14         FOR every poi in pois
15                 poi.addInformationFromSource(source);
16
17 //————————— ADD PHASE ————————//
18 FOR every poi in pois
19         arff.addData(poi);
```

Listing 4: Pseudo-Code explaining the query and data processing pipeline.

A more detailed view is presented in Figure 16, where the first two boxes represent the "QUERY" phase. First, we gather all known instances of noise data, which will be later used to query our other data sources based on each noise datum's location. The details of this phase are covered in Section 4.2.2, where we will present a query adapter for every source. The final "ADD" phase is explained in Sections 4.2.3 and 4.2.4 respectively. In these, we provide ideas on how to convert the queried data into machine learning features and how to combine them into an arff file.

4.2.2 Querying of data sources

Currently, *LOCAL* possesses query adapters for the four mentioned data sources in Chapter 4.1 on page 17. The following section will introduce all of them and explain how the different sources are queried. We also provide some insight into refinements and improvements for selected adapters.

NoiseDataAdapter - Getting initial noise data

The most important source of information and also our initial dataset is the data obtained by the Android app *NoiseMap*. It provides us with a sound level in dB at a location – defined by geographical coordinates – and a corresponding timestamp. As an extra we get to know the id of the used sensor. For easy access the dataset is stored in a SQLite database, whereas the `NoiseDataAdapter` supplies methods to query different subsets of the data. This includes delivering the whole dataset of course, but also subsets including only a defined fraction or a localized subset, such as all data of Darmstadt.

LGDQueryAdapter - Searching for buildings

As explained in Chapter 4.1.3 on page 19, we chose to use the SPARQL endpoint of LGD to extract needed information of entities nearby a given location. These are encapsulated in the class `LGDNode` providing access to its unique node id, its label, the "direct type" marking the most specific type of an entity in the LGD ontology, a list of more general types and of course its location via gps coordinates. Listing 5 shows an example RDF notation (only the description part) of "node344613398".

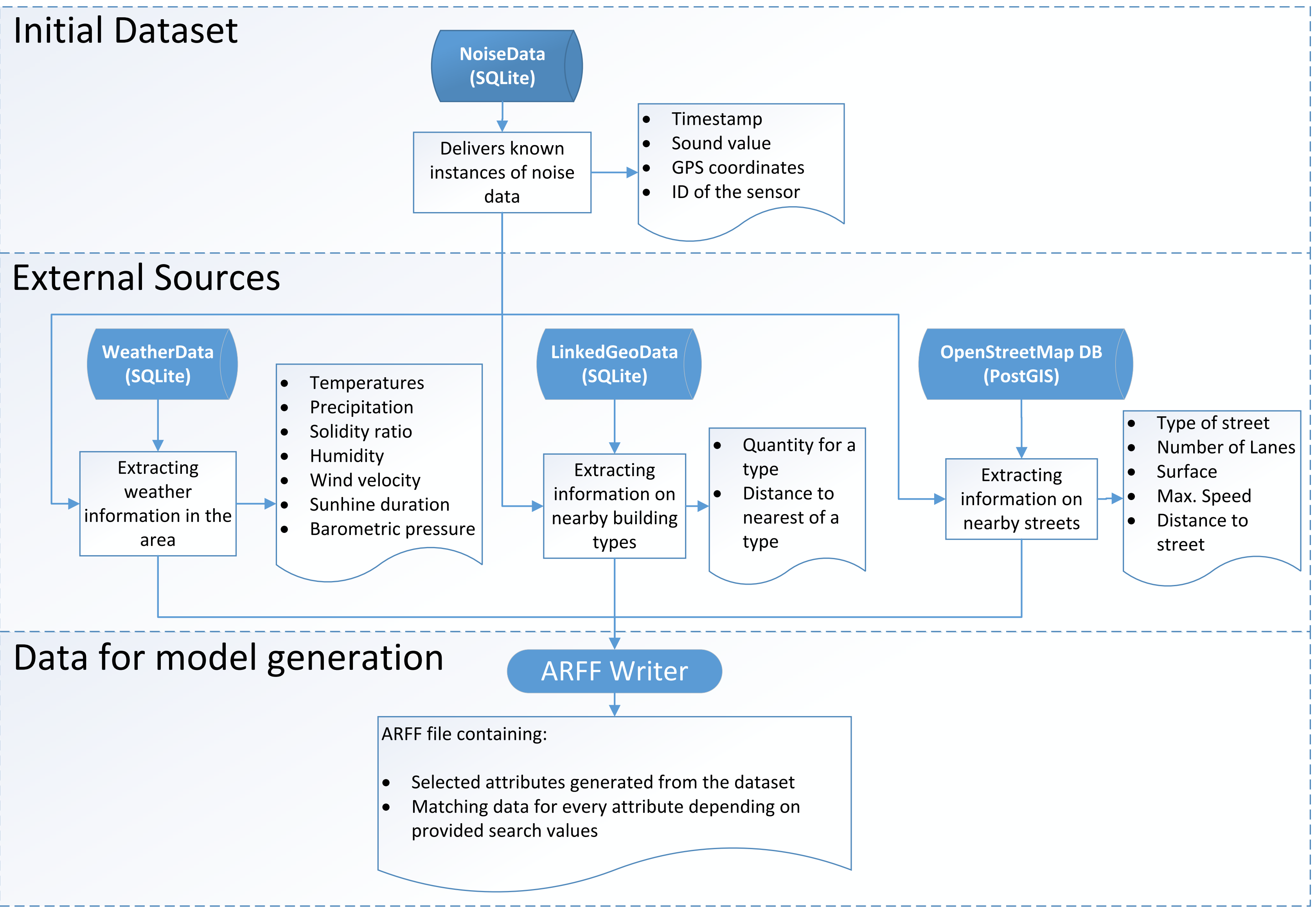


Figure 16: Query and data processing pipeline illustrating the involved databases and the information they provide.

Note the highlighted parts as they contain the information we want to extract. In this case the RDF tells us that the real name of this entity is “Schlosskeller” as stated in line 13. Furthermore we can extract its associated types within the LGD ontology by looking at lines 10 and 12 identifying the facility as a nightclub, which also falls into the amenity category. Last but not least, we can ascertain its geographical location by examining lines 15 and 17.

```
1  <rdf:Description rdf:about="http://linkedgeodata.org/data/triplify/node344613398?output=xml">
2   <rdfs:label>RDF description of Schlosskeller</rdfs:label>
3   <foaf:primaryTopic>
4     <lgdm:Node rdf:about="http://linkedgeodata.org/triplify/node344613398">
5      ...
6       <lgdo:wheelchair rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean">false</lgdo:wheelchair>
7        ...
8       <lgdo:smoking>no</lgdo:smoking>
9        ...
10      <rdf:type rdf:resource="http://linkedgeodata.org/ontology/Amenity"/>
11       ...
12      <rdf:type rdf:resource="http://linkedgeodata.org/ontology/Nightclub"/>
13      <rdfs:label>Schlosskeller</rdfs:label>
14       ...
15      <geo:long rdf:datatype="http://www.w3.org/2001/XMLSchema#double">8.6548286e0</geo:long>
16       ...
17      <geo:lat rdf:datatype="http://www.w3.org/2001/XMLSchema#double">49.8737123e0</geo:lat>
18    </lgdm:Node>
19   </foaf:primaryTopic>
20 </rdf:Description>
```

Listing 5: Excerpt of the description part of the RDF notation from “node344613398”.

As shown in Listing 5, the RDF description provides access to even more information, but since these are specific to certain entities, we restricted ourselves to those properties available for every entity in LGD, which can be done easily by modifying the SELECT statement in the query. Chapter 2.3 introduced the key concept of SPARQL itself. It basically enables us to use SQL-like queries on the LGD dataset. Listing 6 presents an example query to find nearby `LGDNodes` around a given distance from “Schlosskeller” in Darmstadt.

```
1 Select ?node ?label ?type ?directType ?location (<bif:st_distance> (?location, <bif:st_point>
      (8.6548286,49.8737123))) as ?distance) {
2   ?node a lgdo:Node .
3   ?node rdfs:label ?label .
4   ?node rdf:type ?type .
5   ?node lgdo:directType ?directType .
6   ?node geo:geometry ?location .
7  Filter(<bif:st_intersects> (?location, <bif:st_point> (8.6548286,49.8737123), 0.8)) .}
```

Listing 6: SPARQL query searching for buildings in a 0.8km radius around “Schlosskeller”.

Predefined methods, like `st_intersects`, `st_distance` or `st_point`, offer access to geometric properties of entities and allow localized information retrieval. This can be used to further filter the results and reduce their number. In the case of Listing 6, we used them to extract the distance to the found LGD entity as a return value (line 1) as well as to filter out any entities further away than 0.8km (line 7). The remaining lines 2 to 6 correspond to a WHERE clause in SQL. Here, we can assign RDF values to our return variables as well as exploit further WHERE clause capabilities like restricting the query to only specific types in the LGD ontology (line 2). Note that “Node” is the most general type allowing us to retrieve every entity.

Similar to SQL, SPARQL queries produce a `ResultSet`, which can be iterated to convert the results into our internal structure, leaving us with a Map of nearby `LGDNodes` mapped to their real distance to the query location for every noise datum. Listing 7 shows an excerpt of the result of the SPARQL query from Listing 6. As each LGD entity is likely to have more than one type – including super types – we get more than one result line for an entity, e.g. line 2-4 for the “Coffea Bar”.

```
1 "node","label","type","directType","location","distance"
2 ".../node311283855","Coffea Bar",".../Node",".../Cafe","POINT(8.65438 49.8716)",0.2317940402474585
3 ".../node311283855","Coffea Bar",".../Amenity",".../Cafe","POINT(8.65438 49.8716)",0.2317940402474585
4 ".../node311283855","Coffea Bar",".../Cafe",".../Cafe","POINT(8.65438 49.8716)",0.2317940402474585
```

Listing 7: Excerpt of SPARQL result of Listing 6 in csv format. The complete namespace is omitted for better display.

However, we encountered a problem related to the huge amount of noise data, which – as the time of this writing – exceeded 75000 entries. Using the online SPARQL endpoint to query all entries whenever we construct a new arff file was not feasible. To counter this, we established a local database containing the LGDNodes we obtained using all noise data instances as query locations and a maximum distance of 1000 meters. An additional table contains the mapping of every single noise datum to its nearby LGDNodes.

This also provided us with a stable LGD database during the development process and ensured consistent results using various parameters for querying, such as different query distances. As by now, the database used is based on SQLite3, which may not yield the best performance considering the large amount of data. For this reason we provide an interface (LGDAdapter), which defines methods to create and fill a local database and thereby enables the usage of other DBMS, if a better performance may be required.

Furthermore, the LGDQueryAdapter itself, which holds responsible to query a given noise datum for nearby building entities, can be customized by providing an alternative implementation. By doing so the whole pipeline can easily be adjusted to other sources than LGD.

OSMQueryAdapter - Querying for nearby streets

As the SPARQL endpoint of LGD does not provide adequate access to OSM ways, we needed to extract this information out of the OSM data itself. As mentioned in Chapter 4.1.2 on page 18, the data model of OSM consists of three different entities: nodes, ways and relations. Using XML as data format, every entity can be enriched with the corresponding tags and be referenced by its unique id. This is especially important to describe ways and relations, since they relate to a number of nodes. Listing 8 shows an excerpt of an OSM file. Note how the node identifier is used in the way's definition.

```
1  <node id="16541571" version="8" timestamp="2010-03-30T21:35:34Z" uid="6669" user="Elwood" changeset="
       4280528" lat="52.5346714" lon="13.3024201">
2   <tag k="highway" v="motorway_junction"/>
3   <tag k="name" v="Jakob-Kaiser-Platz"/>
4   <tag k="ref" v="3"/>
5  </node>
6
7
8  <way id="4388102" version="7" timestamp="2009-09-22T13:07:19Z" uid="115651" user="Konrad_Aust" changeset
       ="2568991">
9   <nd ref="16541571"/>
10  <nd ref="26750526"/>
11  //more node references
12  <nd ref="26750531"/>
13  <tag k="highway" v="motorway_link"/>
14  <tag k="lanes" v="2"/>
15  <tag k="oneway" v="yes"/>
16 </way>
```

Listing 8: Excerpt of an OSM file showing "node16541571" (with corresponding tags) and its reference in the definition of "way4388102".

Instead of writing our own XML-Parser, we utilize one of the many tools to process OSM data files provided by the community. *Osmosis* is able to convert the XML format into dump files, which can be loaded into a database. As a nice extra, one can specify the construction of bounding boxes and linestring geometries for ways. By using the referenced node ids and their GPS coordinates, the tool asserts the maximum spatial dimension of a way as its bounding box.

Furthermore *Osmosis* provides us with filter capabilities to narrow the resulting database down to ways which are actual roads used by cars or trains in case it is a railway. Listing 9 shows the filter options for the database. Since the OSM keys highway and railway cannot be processed simultaneously, we create two output stream and merge them at the end (line 6). The first stream contains all highways corresponding to the given values (line 1) while the second stream applies the same for all railways (line 2). Note that in line 2 and 5 respectively we remove all relations – as they are not relevant in our case – but advise *Osmosis* to remember all nodes ("used-node") as they are needed to compute the corresponding linestring and bounding boxes of the ways afterwards.

Ultimately we only need the dump files for the ways and way_tags database tables. The latter provides us with all tags specified by contributors and thereby reveals e.g. the street type or maximum speed allowed, while the first table contains the mentioned linestring geometry and bounding box of each way.

```
--tf accept-ways highway=motorway,trunk,primary,secondary,tertiary,living_street,pedestrian,residential
--tf reject-relations --used-node outPipe.0=highway
...
--tf accept-ways railway=light_rail,rail,subway,tram
--tf reject-relations --used-node outPipe.0=railway
--merge inPipe.0=highway inPipe.1=railway
```

Listing 9: Example of an Osmosis filter accepting only the given values for the tags `highway` and `railway`.

These two are spatial attributes, which require a DBMS capable of handling geometry columns and queries. Since *Osmosis* provides dumbs for a *postgreSQL*[43] database, we use the *PostGIS*[44] extension to make full use of geometry columns.

As spatial queries are very compute intensive, it is important to reduce the number of results as early as possible by using the bounding boxes as pre-queries. Consequently, if we search for nearby streets, we do not determine if the linestring geometry intersects with the predefined area around our query location. First, we check the bounding boxes for intersection, which can be done way faster. If there is no overlapping we can safely reject this way and move on to the next one.

Figure 17 illustrates this fact. The blue rectangles represent the bounding box around three given noise data locations, whereas the inner circles represent the "real" search radius specified. The red rectangle is the bounding box of the street "Robert-Schneider-Straße". Note that in the first case there is no intersection between the respective bounding boxes, allowing us to reject the street for location one. This speeds up the querying phase, especially since the indexing of geometric columns allows for efficient checks of bounding box intersections. However, for cases two and three the boxes intersect, which arises the need to check if the actual geometries intersect. Location two's circle does not intersect with the street, which means the "Robert-Schneider-Straße" is not in its defined vicinity. The circle of location three intersects with the street, indicating that it will be considered as a nearby street for this noise datum in the data processing phase.

Listing 10 shows an example query where we included a check for bounding box intersection to speed up the process. In line 3 we check the bounding box of the street – `ways.bbox` – against a constructed bounding box – `Box2D` – around the location of the given noise datum. Note that we us the initial query distance to calculate the vertices of the box. If this evaluates to true, we check the actual geometry boundaries in line 4 by using `ST_DWithin` to ascertain if the linestring geometry of the street lies within the vicinity of the noise datum's location (the blue circle in Figure 17). Ultimately we retrieve the id of the OSM `way`, each associated tag – the key-value pair – and the real distance of this street to the query location (line 1).

```
SELECT ways.id,way_tags.k,way_tags.v,ST_Distance(ways.linestring, ST_GeographyFromText('SRID=4326;POINT
    (8.00863593515 49.9771504215)')) as realDistance
FROM ways LEFT JOIN way_tags ON ways.id=way_tags.way_id
WHERE ways.bbox && Box2D(ST_GeomFromText('SRID=4326;POLYGON((8.007238823820186
    49.97625194890341,8.007238823820186 49.97804889409659,8.010033046479814
    49.97625194890341,8.010033046479814 49.97804889409659,8.007238823820186 49.97625194890341))'))
        AND ST_DWithin(ways.linestring, ST_GeographyFromText('SRID=4326;POINT(8.00863593515
            49.9771504215)'),100,false);
```

Listing 10: Example of a postGIS query using a bounding box check.

The effect of these performance tweaks is quite noticeable. A simple query using only the `ST_DWithin` statement on an OSM dataset of Germany (22645 ways) takes about 85 seconds to complete, the improved one performs over 1000 times faster finishing in approximately 60 ms. Without these enhancement querying OSM data for nearby streets would not be feasible.

Now we can use spatial queries to determine nearby streets around a location within a given radius. Those provide us with the street type – the value of the OSM tag `highway` or `railway` – as well as street attributes tagged by the user. These vary quite a bit for different streets because not every contributor uses each available tag to describe street attributes. Identifying possible attributes with high correlation to the noise level leaves these three: `lanes`[45], `maxspeed` and `surface`.

As a result we introduce a class `Street` containing a mapping of those attribute to their respective value plus its street type and the distance to the query location. Combining multiple `Streets` allows to describe the surroundings of the location. How this can be done will be shown in Chapter 4.2.3, where we will discuss several approaches.

[43] http://www.postgresql.org/ [accessed on 15.05.13]
[44] http://postgis.net/ [accessed on 15.05.13]
[45] Number of lanes

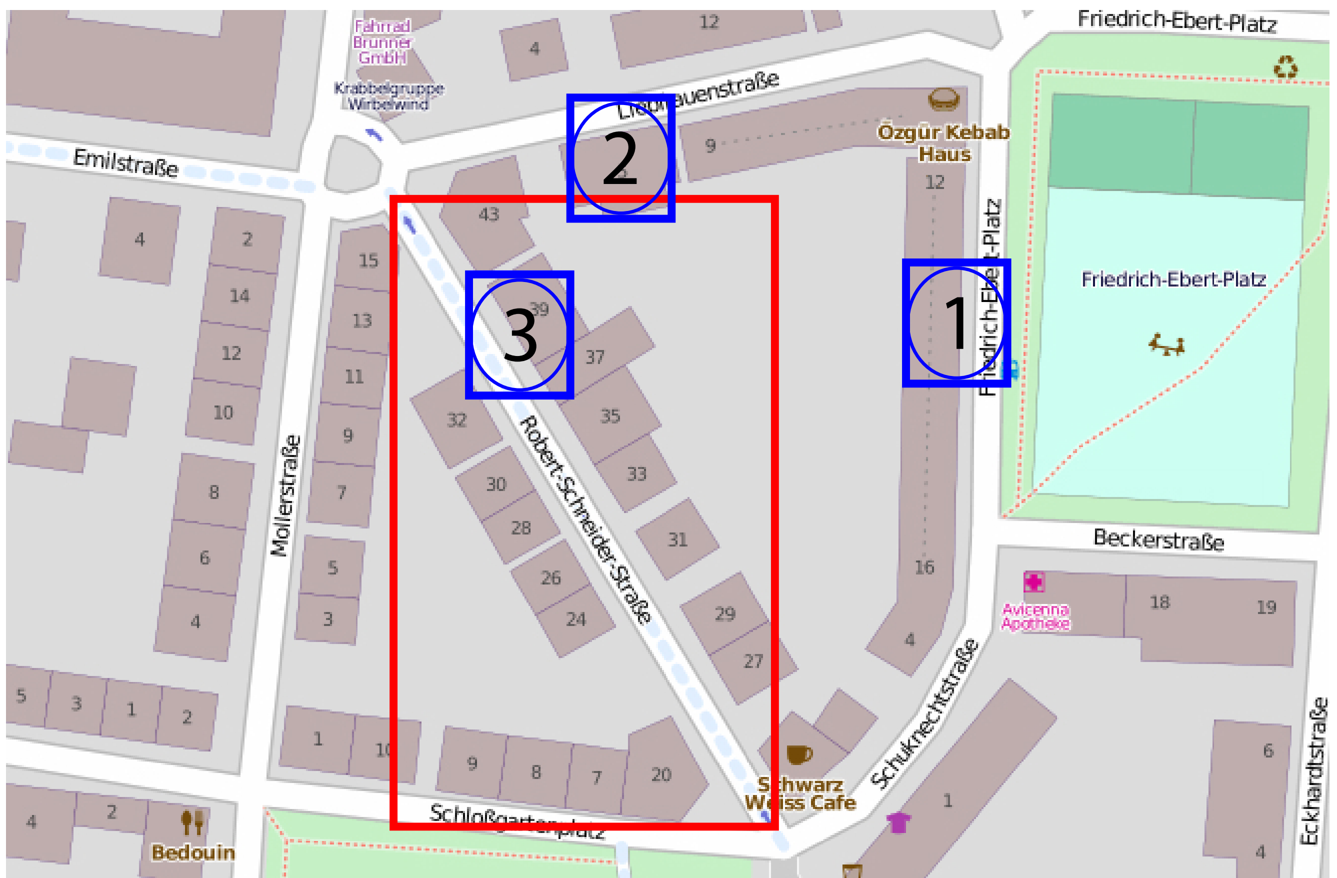


Figure 17: Example of bounding boxes (bbox) usage. The red rectangle marks the bbox of the currently examined street, while the blue rectangles and circles resemble the bbox and real vicinities (based on a radius) of three noise measurement's locations, respectively.

WeatherQueryAdapter - Getting weather information

Contrary to the sources explained before, extracting information about the weather from DWD was a bit more complicated. Free daily data is only available per weather station, represented through two archive files. One contains data for the current year, whereas the other one provides data for all previous years. To be able to query for weather data fast enough, we decided to establish another local database storing information on all available weather stations as well as their respective daily meteorological information.

The job of extracting the relevant data from those archive files is done by the `DwdConnector`. The class provides a single method to update the local weather database by crawling the DWD website. This is done by downloading the archive files for all relevant weather stations, extracting the meteorological values and saving them to the database. We omitted those stations which provide no value to us – like "Zugspitze" or "Helgoland" – leaving 66 stations to work with. The database itself consists of several tables: one holding a list of all suitable stations including their location and unique id, whereas every other table corresponds to one weather station. These list all available meteorological measurements for the respective station sorted by date.

Similar to other sources for which we provided a local database, fetching weather information is done via the `WeatherQueryAdapter`, providing simple access to the database generated beforehand. To receive valid meteorological data, a station must reside within the given query location and the provided maximum distance to this location. By generating a bounding box around the location, we are able to determine which weather stations are possible candidates. Later on we single out the nearest station and provide a `WeatherData` object, encapsulating several meteorologic measurements like sunshine duration, amount of precipitation, temperatures, pressures, etc. recorded by this station.

During our work we also stumbled across other promising sources of weather data like *OpenWeatherMap* as mentioned in Chapter 4.1.4 on page 20. Due to lack of time, this source did not make it into the final implementation, but could be easily adapted. Similar to the previous mentioned sources, we supply the `WeatherQueryAdapter` as an interface to accommodate other possible sources of weather information.

4.2.3 AbstractAttribute - How to model features

For *Weka* to be able to handle the crawled information from our data sources, we need to prepare the data in a preprocessing step. Due to their nature, we need to treat every source differently to extract the necessary attributes suitable for learning with *Weka*. This process is abstracted using the interface `AbstractAttribute`, which encapsulates the preprocessing step for every used feature of our model. Figure 18 illustrate the design, also including all features and other associated classes. By defining only the functionality to add itself to the arff file, every implementation can cope exactly to the needs of each feature – extracted from our data sources – and knows how to handle the required preprocessing steps. In the following we will provide an overview over all implemented features and their composition.

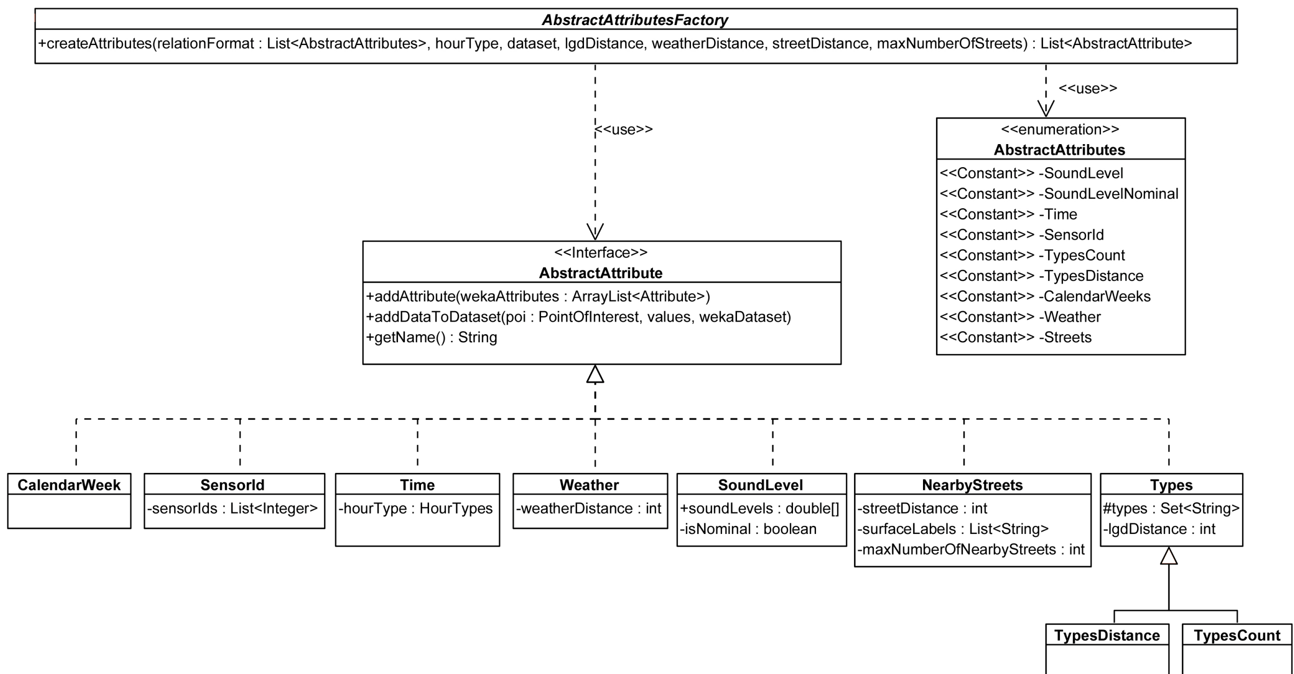


Figure 18: Class diagram of `AbstractAttribute` and associated classes.

SoundLevel

As part of the initial dataset from *NoiseMap*, the sound level itself is rather easy to process. Since we aim to provide a model which predicts a specific loudness range for a given location, the need arises to convert the numeric noise level to an appropriate categorical loudness scale. We will discuss several different approaches for doing so and evaluate their usefulness and feasibility in Chapter 5.1.3.

The most straight-forward method arose from empirical knowledge already known through the work with *NoiseMap*. Taking into account the technical limits of smartphone microphone sensors, the scale ranges from 40-80 dB, divided by equal-width intervals of 10dB. Note that increasing the sound level by 10dB equals to approximately doubling the volume as perceived by the human ear [38]. Adding two intervals for values lower than 40dB and greater than 80dB results in a total of 6 intervals as shown in Figure 19.

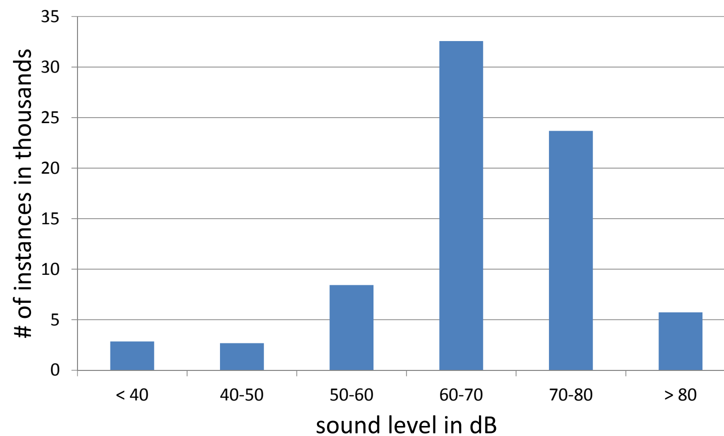


Figure 19: Sound level distribution for equal-width approach.

However, the distribution among those levels is somewhat disproportionate, leaving the range from 60 to 70 dB with almost 44% compared to 4% in the range of 40dB to 50dB. To address this issue, we also evaluate an approach which favors equal-frequency, leading to narrow ranges in the 60 to 80 dB region.

Time

As time is an important factor when it comes to estimating sound levels, the only uncertainty is its granularity. While including seconds seemed to be far too detailed, using only the date as information leaves the important distinction between night and day to be desired. Therefore we experimented with different hourly intervals and eventually settled with one-hourly spans, thus dividing the day into 24 segments as shown in Figure 20. In our opinion, this granularity is optimal to differentiate certain daily noise oscillations while not being to specific, which benefits the classifier.

Building types

One of the main features deals with the buildings surrounding a given location. To model the vicinity of a POI we pursue two different methods. While both rely on the same data provided by LGD, their respective approaches for converting the data to *Weka* attributes is different. As the given query radius around a POI greatly influences both features, finding an optimal distance providing high accuracy while still delivering a good performance is vital and will be discussed in Chapter 5.1.1.

TypesCount: The first approach focuses on the occurrence quantity of nearby building types. For example, a shopping area with lots of cafes and stores greatly influences the noise level. We want to track every building and register their type resulting in a mapping of building types to their number of appearances in the vicinity. If we do not find any facilities of a certain type we set its count to zero.

This is done by collecting all `LGDNodes` in a predefined query range surrounding a POI and examining their types as one `LGDNode` may have several types, ranging from general – like “Amenity” – to more specific ones. Later on we map those to a series of attributes in the weka arff file as illustrated in Listing 11.

TypesDistance: The second approach is very similar to `TypesCount`. Instead of examining the quantity of occurrences of one specific building type, we extract the nearest facility of it and use the distance from the found building to the POI as data value.

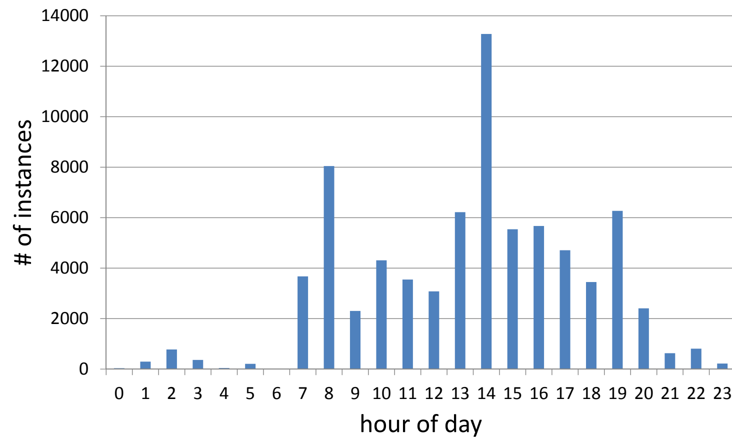


Figure 20: Distribution of all instances for `Time` with one-hourly spans.

```
@attribute BakeryC numeric
@attribute BankC numeric
@attribute BarC numeric
```

Listing 11: Excerpt of an arff file showing header entries of TypesCount.

This does however still require searching for every `LGDNode` within range of the POI. Therefore, limiting the maximum query distance may yield an increase in performance if the accuracy does not decrease.

Worth mentioning here is the fact, that our search for the nearest facility of a specific building type could be in vain. In this case, we could replace the data values with *Weka*'s missing value -– most classifiers can cope with this – or replace it with a value that describes this situation more accurate. Since we started our generation with a fixed maximal distance for the query, we can use this value instead, stating that there is no facility of this type that is nearer than our query distance. To evaluate which method is superior, we will have a look at both of them in Chapter 5.1.1.

Weather

As the name implies this class is responsible for all attributes related to the weather at the POI. In Chapter 4.2.2 we outlined some meteorological measurements we acquired. The complete list includes the following: the sunshine duration and solidity ratio, relative humidity, vapor and barometric pressure, different temperature values such as average, ground, minimal and maximal temperature, average and peak wind velocity as well as precipitation amount and type and if applicable the snow depth. All these values are released under a quality factor, which states if the measurements were recorded automatically or if the recordings were supervised.

Since we do not know how to describe weather in a context that may be suitable for making a connection to the surrounding noise level, we decided to include all measurements and let the classifier itself choose the one with the highest gain.

Due to the straight-forward mapping of weather values to their attributes, adding them to the arff file is not difficult. The only inconvenience related to weather data is the fact that the chosen station may not hold any data for the queried date. In this case, we need to replace it with *Weka*'s missing value.

NearbyStreets

Drawing their information from the OSM database, `NearbyStreets` provides a description of nearby streets in the vicinity of a POI. At first, finding a good mapping of the feature to single attributes was a rather big challenge, since there seemed to be no simple way to map the information of OSM to a format conforming to the arff file. Especially since querying different POI would naturally result in varying quantities of nearby streets. For example, in rural areas only one street may be present while in urban areas there might be much more. Most classifiers do not allow a dynamic approach to describe as many streets as there are in the vicinity. This is due to the fact that they rely on a fixed number of attributes during training and expect the same ones present when classifying unknown instances.

But if we would limit ourselves to a fixed number of nearby streets, we would lose information in case there are more roads nearby, especially in urban environments with their excessive amount of streets. Given the circumstances, we needed a format which would additionally describe the whole area we queried with a limited number of attributes.

To cope with this problem, we create attributes averaging information about all nearby streets. These contain the total count of streets, using the query distance as maximum range, as well as average measurements for maximum speed, number of lanes, and the streets' surfaces, which are all factors greatly influencing the noise level. For maximum speed and number of lanes, using the mean value is straight-forward, however for the streets' surfaces we used the surface who appeared the most among all streets. If we do not possess any information on these measurements, e.g. if they are not tagged within the OSM data, we replace them with *Weka*'s missing value.

Additionally to this summary of nearby streets, we describe a fixed number of roads in detail, including their type, surface, maximum allowed speed and number of lanes as well as the distance of each street to the POI. If there are less streets nearby, we introduce a special street type called "Nothing". Of course we also adjust the surface ("nothing"), maximum speed (0.0), number of lanes (0) and the distance, which defaults to our query distance. By doing so, the used classifier is able to distinguish between the facts, whether there are no further nearby streets or roads with unknown values for maximum speed and number of lanes.

Listing 12 presents the final approach to model `NearbyStreets` in an arff file. Line 1 to 10 represent the detailed description of the two nearest streets while line 11 to 14 provide the attributes for the average values. Note that "..." is used as a place holder for street and surface types to enable a better display.

```
1  @attribute Street0_type {Motorway,Trunk,...,Railway,...,Nothing}
2  @attribute Street0_Surface {asphalt,...,nothing,paved,paving_stones,pebblestone,unpaved}
3  @attribute Street0_MaxSpeed numeric
4  @attribute Street0_NumberOfLanes numeric
5  @attribute Street0_Distance numeric
6  @attribute Street1_type {Motorway,Trunk,...,Railway,...,Nothing}
7  @attribute Street1_Surface {asphalt,...,nothing,paved,paving_stones,pebblestone,unpaved}
8  @attribute Street1_MaxSpeed numeric
9  @attribute Street1_NumberOfLanes numeric
10 @attribute Street1_Distance numeric
11 @attribute NearbyStreets_count numeric
12 @attribute NearbyStreets_maxCountedSurface {asphalt,...,nothing,paved,paving_stones,pebblestone,unpaved}
13 @attribute NearbyStreets_averageMaxSpeed numeric
14 @attribute NearbyStreets_averageNumberOfLanes numeric
```

Listing 12: Excerpt of an arff file showing header entries for `NearbyStreets`.

SensorId

Further exploiting our initial data from *NoiseMap* introduces another aspect which directly influences the sound level. Given the fact that *NoiseMap* collects data from many different smartphones, this somehow introduces "noisy" data since the phones rely on different sensors. To counter this problem, *NoiseMap* uses calibration. Introducing `SensorId` as an attribute helps classifiers to identify "noisy" data, which may have been generated by a badly calibrated device.

CalendarWeek

Prior to adding weather data, we thought of a simple approach to capture the weather in a larger granularity by introducing the calendar week of the year as an attribute. However, after adding proper weather data one might think that `CalendarWeek` is obsolete, but only if you restrict your point of view on the weather itself. Considering certain events, which take place at a fixed time or week in the year, `CalendarWeek` may hold clues explaining abnormalities in the registered sound level, e.g. a sudden rise in volume due to a huge street festival, like the "Schloßgrabenfest" in Darmstadt. If that theory holds true will be discussed in Chapter 5.1.1.

4.2.4 DataBuilders - Generating a custom arff file

The first two sections of this chapter provided an overview over adapters used to query data sources as well as how we model the crawled data into attributes suitable for machine learning.

However, we need to make sure to use the right query adapter for the right feature. To ease this process, we introduce a variety of `DataBuilders`. These encapsulate the procedure of selecting query adapters based on given features. Each `DataBuilder` can be decorated with another `DataBuilder` resulting in consecutive querying of the appropriate sources during runtime.

The composition is done by the `POIBuilderFactory`, which relates a list of enumeration constants (`AbstractAttributes` see Fig. 18 on page 27) representing our features to their respective `DataBuilder`. This allows for a good abstraction over the querying and refinement process of the various data sources. Figure 21 shows associated classes, including all

`AuxiliaryBuilders` and the initial `POIBuilder`, which is used to generate a list of POIs out of the given list of noise data. Note that the `queryData` method is responsible for firstly querying the appropriate source by using a query adapter, and secondly for modifying the entries of the POI instance accordingly.

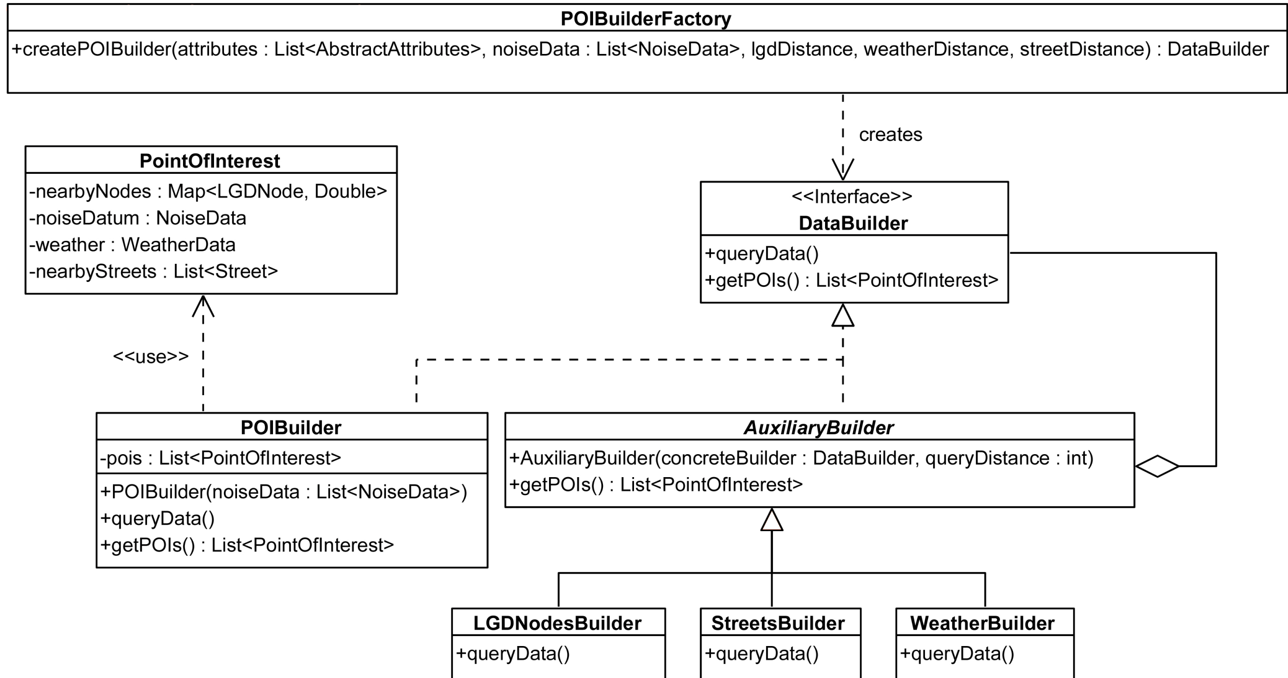


Figure 21: Class diagram of `DataBuilder` and associated classes.

The transition from our internal dataset – the list of POIs – to the appropriate *Weka* classes is done by the `InstancesGenerator`. It combines the process of querying all `DataBuilders` and then generates *Weka* compatible instances by mapping the retrieved data. Since this approach requires querying all data at first and be present in memory, it is not suitable for huge amounts of noise data as well as large query distances.

To address this issue, we introduce the `InstancesChunkGenerator` which keeps the memory overhead to a minimum, but lacking in performance. The key idea is to process the data in smaller chunks. Due to the fact that we need to have a complete overview over all instances to assign the used *Weka* attributes, we need to scan the whole dataset for two times. As an example, recall our internal `TypesCount` feature. It encapsulates every building type we found into several *Weka* attributes, one for each type. This means that the number of needed attributes dynamically rises if we increase either the number of noise data or the maximum query range, as we are prone to find more unique types. A pre-scan of the complete dataset resolves this issue by determining which attributes need to be added. After this, we can assign values by processing the dataset a second time.

The final output file is generated by the `ArffFileWriter`. It takes a set of instances from the `InstancesGenerator` and converts them into an arff file, which can be loaded into *Weka* itself for classification. As the handling of the `InstancesChunkGenerator` is somewhat different, the `ArffFileWriter` takes the responsibility of communicating with the generator itself to request the appropriate data chunks when needed.

5 Evaluation

Successful data mining does not only include choosing the "right" classifier for a given problem. There are additional aspects which need to be considered. On the one hand, most classifiers allow fine grained adjustment of various parameters like pruning or splitting factor in decision tree learning or adjusting the value of k in a k-nearest-neighbor classifier [44].

At the other hand, data transformation cannot be overlooked if we want to optimize our results. This includes attribute selection, data cleaning or sampling. While the latter two aim to reduce noise in the dataset itself, feature selection produces a subset of attributes by removing redundant, clearly irrelevant or even contradictory ones. This may speed up performance as well as increase accuracy. [44]

In the following chapter, we will have a detailed look at feature selection and refinement as part of our evaluation mentioned in Chapter 4. We believe that this method will yield better results than a parameter refinement, as a thorough evaluation of the latter would not be feasible given the huge dataset and the limited amount of time.

We start with optimizing single features on their own in Section 5.1.1 and follow up with an evaluation of several feature compositions. In the next sections (5.1.3 and 5.1.4), we will discuss the influence of different sound level distributions as well as different classification algorithms. Section 5.2 will sum up our results as well as provide a conclusion.

5.1 Feature selection and optimization

Experiments in literature show that adding random binary attributes to a dataset which was then learned by a decision tree learner decreased its prediction accuracy by 5 to 10% [44]. In theory this should not happen, since the classifier should always choose the attribute with maximum gain. Although this holds true at first, as we proceed further down a decision tree, the amount of data on which the classifier bases its decision reduces, allowing random attribute values to come into play. [44]

The example shows that feature selection can improve accuracy. As it is not feasible for us to use a feature selection algorithm based on a search through the attribute space itself due to the fact that our dataset contains a huge amount of instances and attributes, we decided to manually evaluate the benefits of the features presented in Chapter 4.2.3 on page 27 on their own as well as in composition with each other.

First, we look at each feature in isolation to get a first indication for its general usefulness. The next step combines several features and evaluates their cumulative accuracy. We use the knowledge originating from the first two steps to construct an optimal combination of features to maximize the accuracy when predicting noise levels.

Each evaluation is performed in *Weka* using a 10 fold cross validation and J48 as a classifier. A more detailed reason why we chose J48 is presented in Subsection 5.1.4 where we also look at the influence of several other classifiers.

5.1.1 Single features

This section focuses on evaluating each feature in isolation. In particular, we compare different approaches we introduced in Chapter 4.2.3 to model the presented features as well as finding an optimal query distance for those that require one. By doing so, we can ascertain an optimal representation of each feature, which benefits the following evaluation steps. As the `SoundLevel` is our class attribute and thereby a special case, we will evaluate different distributions in a following section. For now we will use the equal-width approach – presented in Chapter 4.2.3 – for the evaluation of other features.

As a baseline we use a model that only has access to `Time` as well as the class value `SoundLevel`, which we want to predict. Therefore `Time` is the only attribute which can be used by the classifier to determine a noise level. The class attribute provides information on whether the prediction was correct or incorrect. For every contemplated feature we provide accuracy results for the baseline plus the feature itself, meaning every evaluation already includes the baseline.

SensorId and CalendarWeek

Both features include no parameter to adjust their results. Therefore the evaluation of these is quite simple involving only one run each. Figure 22 shows how well the two features are predicting a noise level in comparison to the baseline.

As both features provide additional knowledge their accuracy is naturally higher. However, as we mentioned in Chapter 4.2.3 on page 30, *NoiseMap* provides us with calibrated sound values. This means that there should be no significant difference when looking at noise values from different sensors. Although a gap of approximately 15% may indicate otherwise, this can also be caused by contributors living in a noisy environment which makes their sound measurements prone to be higher than normal.

If we look at `CalendarWeek`, the gain is not as high as with `SensorId`. Nevertheless it does benefit the achievable accuracy. There are numerous possible reasons why this is the case. We already addressed two possible factors in Chapter 4.2.3 on page 30, namely coarse weather information and weekly events. Yet we have no real evidence supporting these theories, as that would include examining the instances one by one for such events, which is just not feasible given the amount of data.

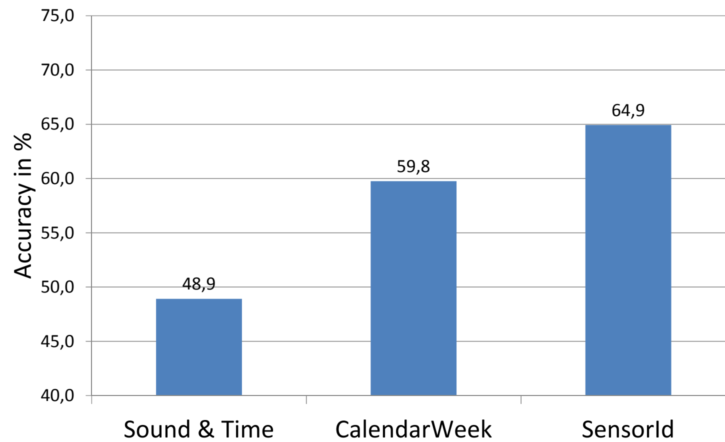


Figure 22: Accuracy of the baseline, `CalendarWeek` and `SensorId`.

Weather

Clearly outmatching the previously examined features, `Weather` does boost prediction accuracy up to 70.4%. Due to the fact that we prefer to use the nearest possible weather station for data, we only need to evaluate the minimal range at which we get weather information for every noise data. Further incrementing is counterproductive as the result does not change – the nearest station would still be closest – but computational costs would increase. Figure 23 illustrates this evaluation process, as we begin to increase the query distance successively. Note that after 60km there is no further increase in accuracy.

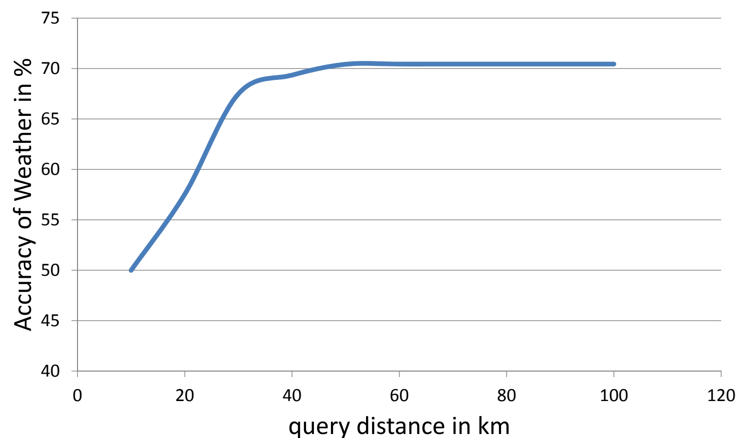


Figure 23: Accuracy of `Weather` for increasing the query distance.

NearbyStreets

As we explained in Chapter 4.2.3 on page 29, there are several ways to describe the vicinity of a POI when it comes to nearby roads. In this section we compare possible approaches we touched beforehand.

Since there are various unknown parameters – equaling the amount of different approaches – we need to adjust one of those during an evaluation run, while fixating all the others. An exploration of all possible combinations would be too time consuming, which is why we decided to try an iterative approach by optimizing one parameter at a time and use the acquired knowledge henceforward.

First of all, we evaluate the query distance which achieves the highest prediction accuracy. We start this off by describing the closest two streets explicitly plus the group of attributes we used to model the overall vicinity of a POI. As a reminder, this group includes the total count of nearby roads as well as averages for maximum speed and number of lanes followed by naming the surface which appeared most. From this point on, we will call it the averages group. Figure 24 illustrates how the accuracy develops when increasing the query distance.

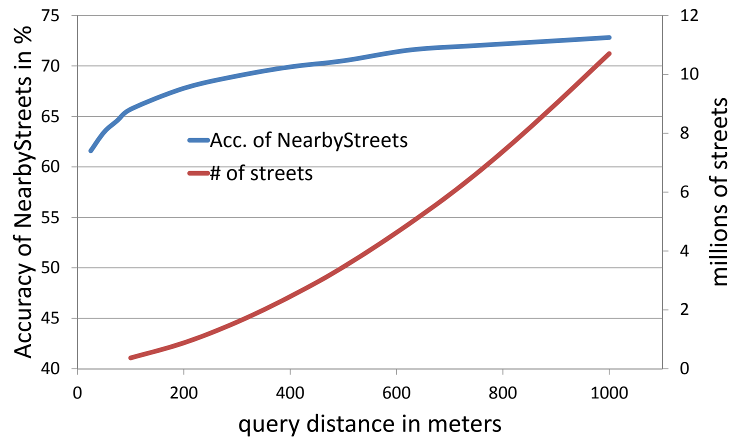


Figure 24: Accuracy of `NearbyStreets` for increasing the query distance against the processed number of streets.

For small distances the increase is significant, while slowing down at approximately the 200m mark. If we recall our attribute composition, this could be due to the fact that only the group of averages benefits from further raising the query distance, as the closest two streets stay the same. Although we can educe from the diagram that incrementing the distance past 1000m would result in an even higher prediction accuracy, this would drastically increase the quantity of streets processed and thereby also rising the memory overhead during query time, leading to a bad ratio of improvement to memory overhead. As this was not feasible with the given equipment, we limited ourselves to a maximum range of 1000m.

Using an optimal query distance of 1km, we can now further evaluate other parameters such as the number of streets we describe in detail. Not knowing how this may be influenced by the query distance itself, we provide results for three different query distances to be able to spot possible irregularities. Figure 25 illustrates the accuracy for using zero to five closest streets described in detail. Keep in mind that zero nearby roads equals to only having the group of averages.

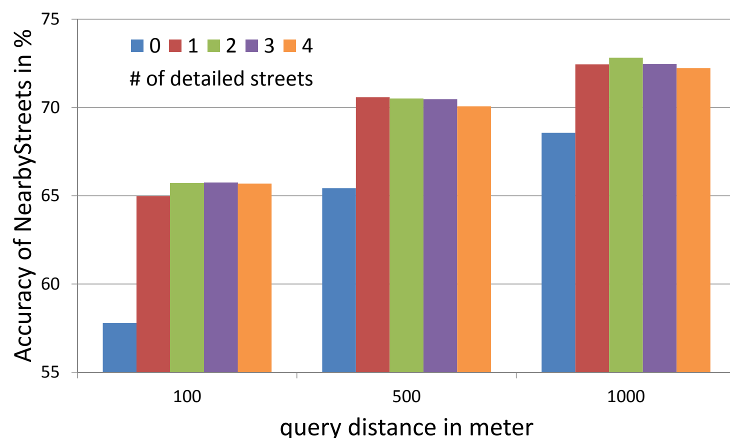


Figure 25: Accuracy of `NearbyStreets` for three query distances with varying number of fully described nearby streets.

Explicitly describing a number of closest streets drastically increases the prediction accuracy throughout all displayed distances. However, the count itself has very little influence on the result. Though using two streets takes the lead for the previously evaluated optimal distance of 1000m, adding three or only one streets yield the best results for 100m and 500m respectively. As the gap is negligible, we favor detailed specification for the two closest roads resulting in an overall maximal achievable accuracy of ca. 73%.

Now that we have acquired a coarse grained parameter setting, we can proceed with further tuning. Recall the case if we want to describe the two closest streets but we only have one street nearby. In our initial approach – discussed in Chapter 4.2.3 on page 29 – we add a “Nothing” street. Alternatively, we could have used *Weka*’s missing value. Diagram 26 shows the two alternatives in direct comparison utilizing the two closest streets and three distinctive query distances.

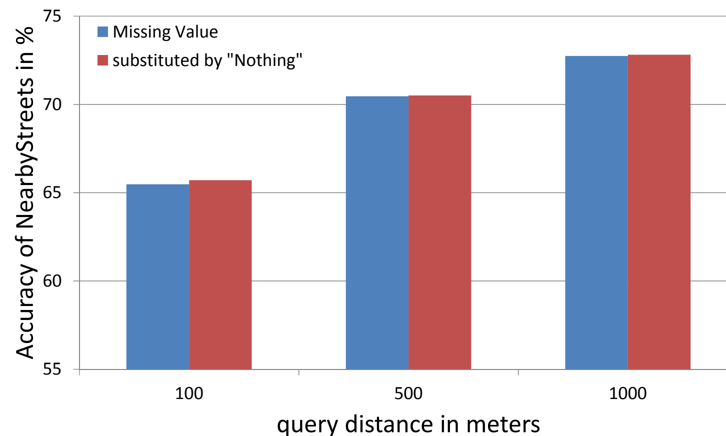


Figure 26: Comparison of using “Nothing” street against missing value for three query distances.

Since we can acquire more than two nearby streets for most POIs, the gap between the two approaches is rather slim with “Nothing” having a slight edge. As we mainly use the nearest two streets, we did not evaluate other street counts. However, we expect the gap to increase with increasing number of nearby streets described. This is due to the fact that “Nothing” actually is a more valuable information for the classifier, as it is one defined value compared to a missing value, which can be an arbitrary one out of the given possibilities.

As mentioned early in this chapter, we went for an iterative approach to determine optimal parameters, which e.g. resulted in assuming that the group of averages would improve the estimation accuracy. To address this issue, we made another evaluation comparing our optimal approach, consisting of the two closest streets and the group of averages, with an approach relying solely on the two nearest roads. This comparison is shown in Figure 27.

Using only the two closest streets does not provide further information when increasing the query distance, stagnating at approximately 65%. Describing the whole vicinity with a group of attributes on the other hand significantly boosts the resulting accuracy when increasing the query distance. This confirms the initial theory outlined in Chapter 4.2.3 on page 29 to be useful.

TypesCount

Another more straight-forward feature is `TypesCount`. Since there is not much to refine how to convert this feature into *Weka* attributes, we can focus on finding the optimal query distance. Figure 28 shows the accuracy trend from a minimum query distance of 20m to a maximum of 1000m as well as the processed number of `LGDNodes` as an indicator for computational and memory load.

Compared to `NearbyStreets` the incline is more linear starting at approximately 50% for 20m and reaching a maximum of 72.2% for 1000m. The more linear fashion – compared to `NearbyStreets` – could indicate a greater potential for improvement past the 1000m mark. Likewise, our equipment as well as the LGD server itself imposed this limitation upon us. Nevertheless, the steady increase up until the maximum mark is actually quite remarkable. Putting it another way, this means that even buildings at a 1km distance may influence the noise level at a given location.

One possible cause for this phenomenon could be the influence of “Fingerprinting”. This kind of technique is frequently used to identify a person’s location by using nearby WLAN access points [45]. If we map this approach to our dataset we can create a fingerprint of a noise data instance by mapping building types and their occurrence. Slightly moving the location would then result in only a minor adjustment of the fingerprint and therefore estimating a similar noise level.

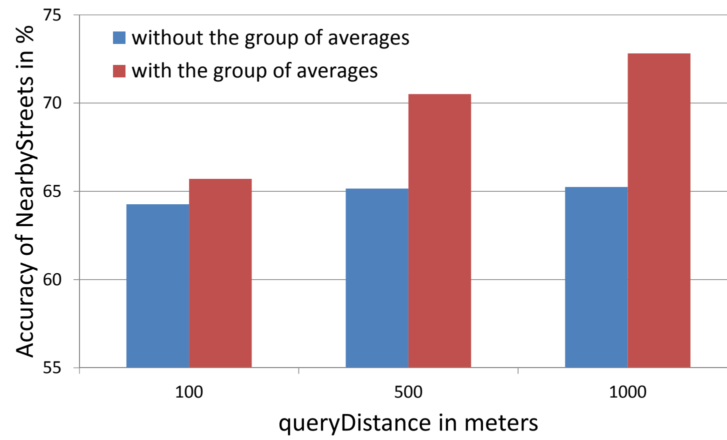


Figure 27: Accuracy of `NearbyStreets` with and without the group of averages for three query distances.

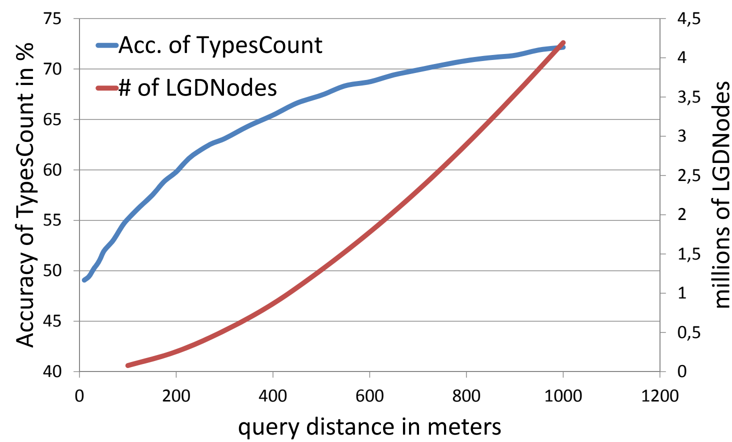


Figure 28: Accuracy of `TypesCount` for increasing the query distance against the number of processed `LGDNodes`.

Remember that the classifier itself does not know the geographical coordinates of a location. On the other hand a big difference between fingerprints would indicate a different sound level as well as big distance between the two locations.

Although this may explain the accuracy trend, there is no evidence to back it up. There could very well be a real influence of certain building types towards the noise level, e.g. airports with their wide range of noise pollution.

TypesDistance

The evaluation for `TypesDistance` is quite similar to `TypesCount` as both use the same data source, but just interpret the data differently. Additionally to finding an optimal query distance, we want to evaluate if replacing the attribute value with the query distance itself – in case we do not find a facility of the current type – yields better results than using *Weka*'s missing value. Both accuracy gradients are illustrated in Figure 29 as well as the processed number of `LGDNodes`.

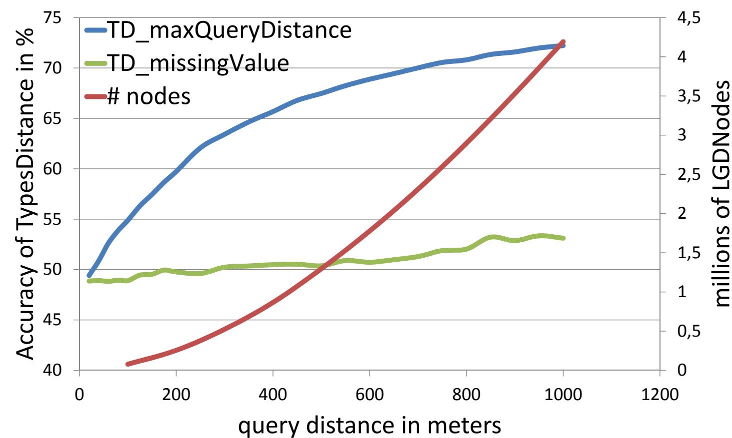


Figure 29: Accuracy of `TypesDistance` (missing value and max query distance) for increasing the query radius against the number of processed `LGDNodes`.

The first thing which meets the eye is the fact that replacing missing distance values with the query radius is far superior compared to utilizing *Weka*'s missing value. The latter almost stagnates at approximately 50%, while the first one shows a nearly linear increase from 50% to 72.2% at 20m and 1000m respectively. As explained in the section about `TypesCount`, this may be influenced by several causes. The fact that both type features bare very similar appearances concerning their respective diagrams will be discussed in Chapter 5.1.2, as this is part of the evaluation covering different feature compositions.

Summary

As a conclusion, Figure 30 shows an overview of all features and their maximum achievable prediction accuracy. Leading the chart are the two building type features `TypesCount` (TC) and `TypesDistance` (TD) as well as `NearbyStreets` (Str) with approximately 72% prediction accuracy, followed by `Weather` (W) with ca. 70% and `SensorId` (Id) with 65%. The last place goes to `CalendarWeek` (CW) achieving only 60% accuracy. The first column represents the baseline using only `SoundLevel` and `Time` (S&T) with almost 50%.

While this provides a first impression of the benefits of each feature, the next chapter will focus on different composition of those features, evaluating which combination yields the highest accuracy. We also have a look at possible redundant features which may decrease performance and accuracy.

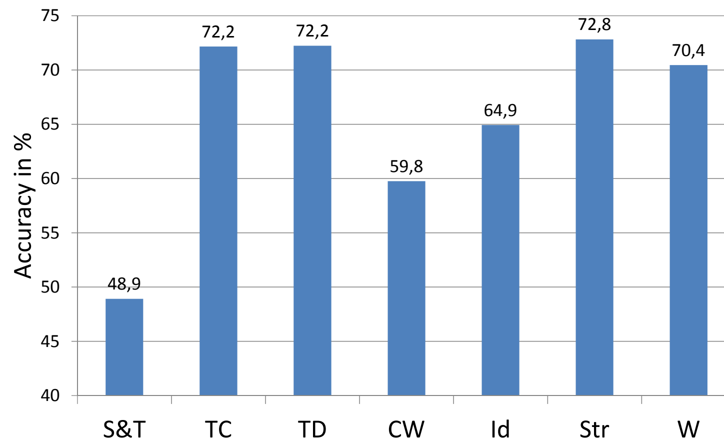


Figure 30: Maximum achievable accuracy of each feature compared to the baseline.

5.1.2 Feature compositions

The previous chapter explained the benefits of every feature in isolation, while this chapter will focus on uniting features to generate a higher overall prediction accuracy. During our search for noise factors, it was not possible to estimate the correlation between these. To illustrate this, consider a simplified version of the features `Weather` and `TypesDistance`. If we look at them in isolation, a rainy day may indicate a lower noise level, since outdoor activities are not that attractive anymore. On the other hand, at a location prone to have a low noise level, the real level could actually be higher, since the rain itself produces a good amount of noise. Having only weather information makes it very hard to differentiate between these two cases. Adding `TypesDistance` on the other hand could tell us, whether we are inside a park or in the city center.

Coming back to our example with the rainy day, we can now assume a lower noise level for the park, since this location is mostly noisy on sunny days, as it gets used for outdoor activities. In the city center most noise comes from the traffic. A wet street could thereby further increase the noise level. In this case, we would assume an increased accuracy when combing both features as they support each other. However, this may not always be the case. Redundant or contradictory features can decrease the prediction performance.

Our main concern is thereby to find out if features conflict with each other. Additionally, we want to remove redundant features, as this would increase the efficiency of our approach.

Since we created six different features, apart from `SoundLevel` and `Time`, this would require quite a few evaluation runs. Therefore we cannot evaluate every possible combination. The following diagrams show the accuracy results when consecutively adding the features from left to right.

Figure 31 starts of with the two LGD features – `TypesCount` and `TypesDistance` – followed by adding `Weather`, `CalendarWeek`, `SensorId` and lastly `NearbyStreets`. Surprisingly, adding `TypesDistance` does increase accuracy only by a mere 0.25%. As mentioned before, redundant attributes do not provide further information usable for classification. Comparing the format of the LGD features, we can deduce that both actually describe the same information although the representation is slightly different.

To illustrate this fact, consider the following example. In a location with no facility of a specific building type around, `TypesCount` would provide a value of 0, whereas `TypesDistance` would value to 1000. This means that asking if CafeC == 0 is equivalent to CafeD == 1000. However, this alone is not sufficient to ascertain a redundancy between the two features. They need to additionally provide the "same" values if there are buildings nearby. Even though they do not provide identical ones, those values are inverse proportional, meaning that they increase for `TypesCount` and decrease for `TypesDistance` if facilities of building types are found. Even more, for a high value in `TypesCount` the corresponding one in `TypesDistance` is prone to be rather small, since it is more likely that a facility of this type will be very close.

Omitting one of the two LGD features can increase performance when creating a decision tree and also during classification. However, it does not speed up the initial query phase, since both features rely on the same data. If classification performance is a key factor, omitting one of those features speeds up classification while not loosing much accuracy. But since predicting the class value for a test instance is rather fast on decision trees anyway, having a slightly higher accuracy may in fact be more favorable.

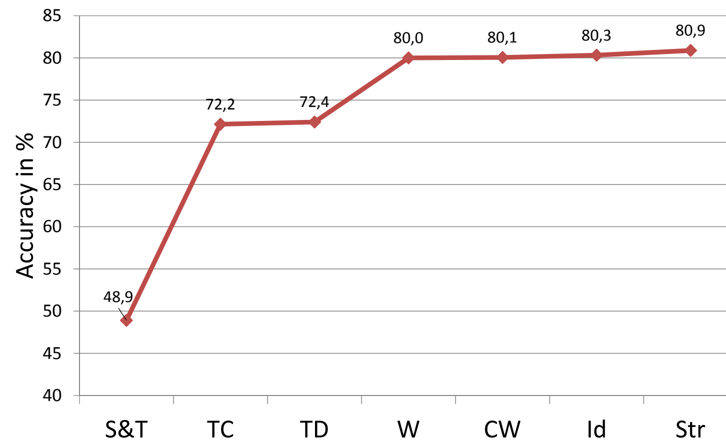


Figure 31: Accuracy of the given cumulative combination of features.

Another interesting fact is that using only `TypesCount`, `TypesDistance` and `Weather` we are able to achieve approximately 80% accuracy. Further features only increase this result by a margin. This, however, is to be expected. If we start of by predicting the majority class of our six different noise levels, we can achieve circa 43% accuracy. Adding little knowledge in form of additional data as well as using an elaborated classifier drastically boosts the results, as our state changes from not knowing anything to knowing a little. Since we already have a huge knowledge with the three features combined, adding further knowledge does not result in such a huge impact. Although an increase of only one percent point may not sound that much, it is much harder to achieve at already 80% accuracy than at 50%.

Based on this, we cannot ascertain a correlation between features past adding `Weather`. To address this issue, we run additional evaluations where we change the order of adding features.

In the second run, features that achieved less accuracy on their own get added first. Starting with `CalendarWeek` and `Weather`, followed by `SensorId` and `NearbyStreets`, we add the LGD features at the end this time.

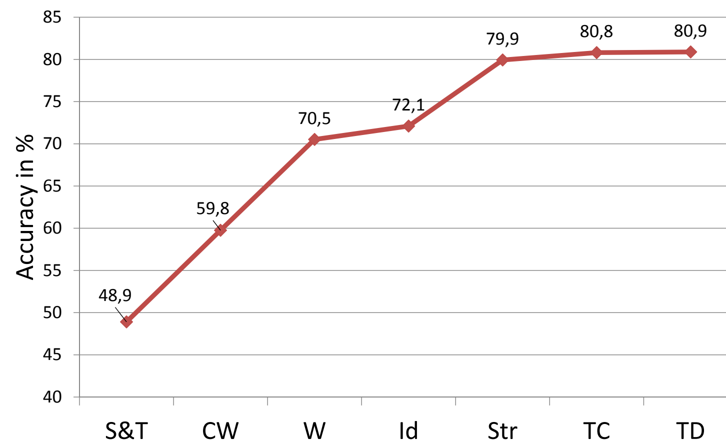


Figure 32: Accuracy of the given cumulative combination of features.

In Chapter 4.2.3 on page 30, we discussed similarities between `CalendarWeek` and `Weather`, as we introduced the first as a form of a coarse grained weather forecast. Looking at Figure 32, we can deduce that the detailed weather information is far more valuable. Although `Weather` significantly increases accuracy, based on Figure 32 alone we cannot conclude that `CalendarWeek` is redundant, if we also add `Weather` as a feature. However, if we compare the accuracy

results achieved by `Weather` alone and in combination with `CalendarWeek`, we achieve 70.4% and 70.5% respectively. Based on this observation, we may omit `CalendarWeek` as a feature if we want to build a small yet accurate model. As it is only a small feature, which does not require querying an extra data source, removing it is only feasible if we really want to have the smallest possible model.

After adding `NearbyStreets`, we reach an accuracy of approximately 80%. As explained before, drawing conclusions from accuracy gain at this stage is not feasible. This can also be clearly seen by the fact that adding one of the best features `Typescount` only increases it by one percent point.

Our final run presented in this chapter includes `NearbyStreets` as first feature, followed by `CalendarWeek`, `SensorId` and `Weather`, finishing with `TypesCount` and `TypesDistance`.

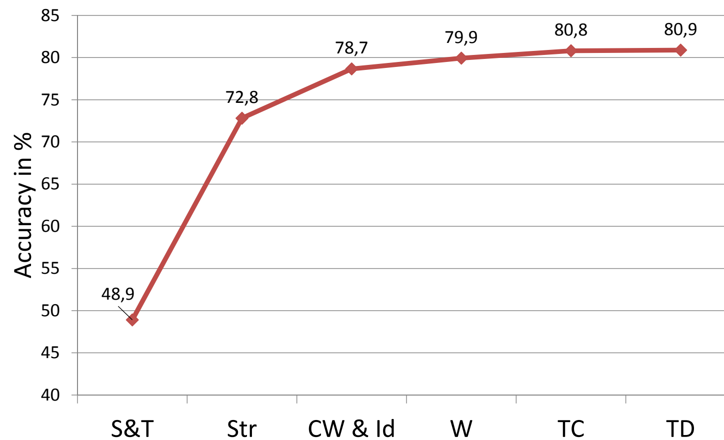


Figure 33: Accuracy of the given cumulative combination of features.

Since `NearbyStreets` provides a high information gain on its own, adding `CalendarWeek` and `SensorId` – the two weakest features – increases accuracy by only a small amount as shown in Figure 33. At this point we almost reach the optimal accuracy, which means that adding `Weather` and the LGD features only provide small increase. This shows that choosing the right features to construct a small model, that yields the highest possible accuracy, is important, since adding weak features, that get outmatched by stronger ones, is not suitable.

As a conclusion, the evaluation of different feature compositions provided us with a way to identify weaker features on a more solid basis, since it takes feature compositions into account. We evaluated that using all features results in the highest prediction accuracy (80.9%), but leads to a rather large model, which slows down the performance.

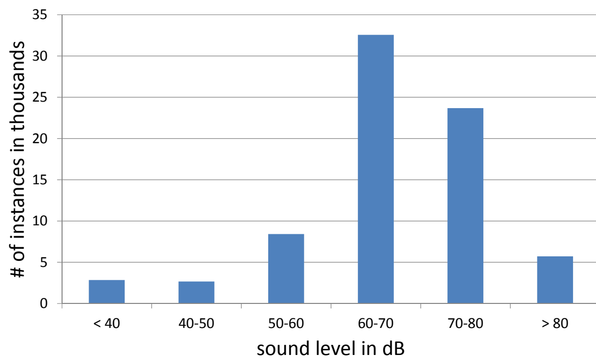
In case a small model is more beneficial, we learned that removing partly redundant features can drastically reduce model size, while only sacrificing little accuracy. This particularly holds true for the two LGD features, as they consist of many *Weka* attributes and thereby are especially responsible for a huge model. As shown in Figure 33 on page 40 using `NearbyStreets`, `CalendarWeek`, `SensorId` and `Weather` already results in a prediction accuracy of almost 80%, while producing a rather small model.

But there may also be other factors which influence such a decision, e.g. the availability of data sources for a certain area. If we do not have access to weather data, we cannot use the feature. Since we only have weather information for Germany, this feature would be unusable for foreign countries.

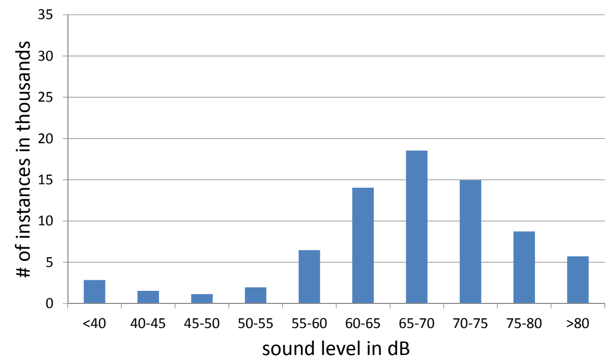
As we did not evaluate the individual *Weka* attributes of each feature – simply because of the huge training set – we cannot ascertain redundant or contradictory attributes within particular features.

5.1.3 Different sound level distributions

Apart from different approaches in modeling our features, we will also have a look at varying distributions for our class attribute as presented in Chapter 4.2.3. In particular, we have a look at how the distribution affects prediction accuracy and can be modified to our advantage by evaluating two equal-width as well as two equal-frequency distributions. Figure 34 shows the former approach by using equal-width intervals ranging from 40dB to 80dB, whereas we used six distinctive intervals in Diagram 34a compared to ten in Figure 34b. Note how this distribution favors the range between 60dB and 70dB resulting in a rather high accuracy if we would only predict the largest class.



(a) Six different sound levels.



(b) Ten different sound levels.

Figure 34: Sound level distributions for equal-width approach.

To counter this issue, we also evaluated equal-frequency distributions with six and ten sound levels respectively. The former one can be seen in Figure 35 where the interval borders are marked with red bars. Figure 36 shows the same approach but with ten intervals instead of six. Both clearly illustrate the large sample count in the 60dB to 70dB region, as the distance between each border becomes very small. Although we have accomplished our aim of an uniform distribution, it is now more difficult to distinguish between intervals, as their range is considerably smaller in regions with a high sample count, which may lead to a decrease in prediction accuracy.

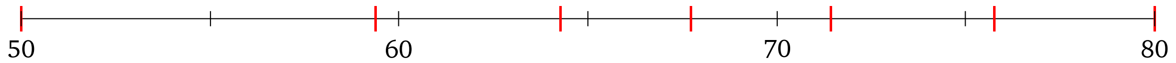


Figure 35: Equal-frequency approach with six intervals.

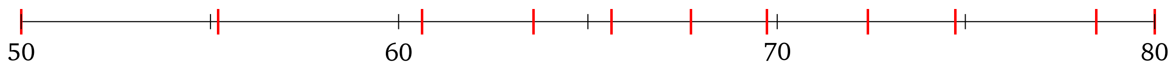
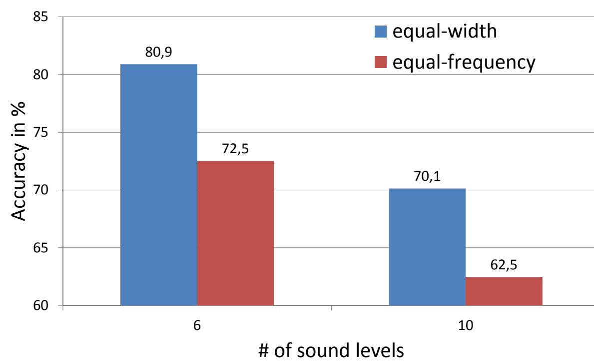


Figure 36: Equal-frequency approach with ten intervals.

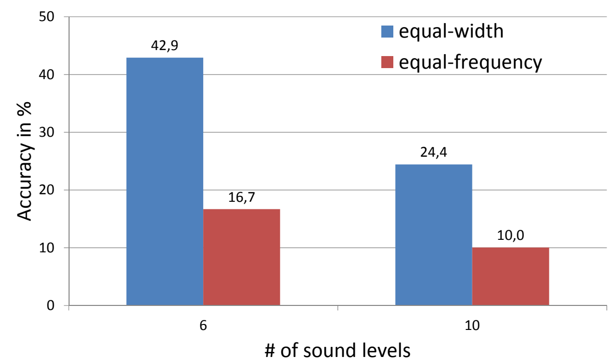
To identify the optimal distribution of sound levels, we constructed models for each approach using all our features with their optimal parameters and J48 as classifier. As a baseline we predict the majority class (ZeroR classifier) of each distribution indicating an a priori prediction accuracy. Our results are presented in Diagram 37 indicating a clear win in terms of accuracy for equal-width with six intervals. Looking at the baseline results (Figure 37b) compared to the one of J48 (Figure 37a), we notice a high potential for accuracy improvement for the equal-frequency method. However, this is to be expected, as each sound level holds exactly the same amount of samples, resulting in an indirect proportionality between the number of classes and ZeroR's prediction accuracy. Although this trend can be seen for the equal-width approach as well, it is less distinctive.

In general using only six sound levels yields a ten percentage points higher accuracy for both approaches when using J48. However, this may seem deceiving. On the one hand, we increase our prediction accuracy, but on the other hand we also increase the range of all sound levels. For example, if we use only one noise level, our prediction accuracy would be 100%, but we would not have gained anything by doing so. Considering this, we cannot ascertain a clear victorious method for distributing the noise levels, as this highly depends on the needs of the application. Using equal-width, we can predict a noise level within a 10dB range with over 80% accuracy or alternatively within a 5dB range with approximately 70%, which may be more beneficial for some use cases.

Although equal-frequency is always inferior to the equal-width approach in terms of accuracy, it may yield great benefits for applications demanding very fine-grained interval steps in the region around 70dB. To put it another way, we achieve highest accuracy in regions with high sample count – resulting in small interval ranges – while sacrificing it in sparse areas (around 40dB). This provides a more dynamical approach towards sound level distribution, while equal-width is more static.



(a) Accuracy results for J48.



(b) Accuracy results for baseline (ZeroR).

Figure 37: Accuracy of different distributions. Be aware of the different axis scales.

As outlined before, the ideal distribution strongly depends on the applications demands and should be adjusted accordingly to provide best results. In Chapter 6 we discuss an interactive heatmap approach making use of the equal-frequency scheme to emphasize noise changes.

5.1.4 Different classifiers

Something we have not touched yet is the influence of the chosen classifier on the resulting accuracy. Since our dataset includes numeric and categorical attributes as well as missing values, we need an elaborate classifier which can cope with these "real-world" problems. We covered the world of machine learning and different classifier in Chapter 2.2 on page 5.

In this section, we will focus on efficiency and practicality of the classifier during the creation and classification phase. Classifiers we can omit from the start are instance-based approaches. They may be fast with regard to the creation phase, as no model needs to be established at all, but they severely lack when it comes to classification speed. Since our dataset contains 75000 instances, running a nearest-neighbor approach is not feasible. This is also a KO criterion for ensemble methods, as they involve several classification stages resulting in low performance.

Remaining types would include decision tree learners, rule set learner and support vector machines beside many more. We also evaluated one "meta" classifier using ordinal classification and J48 as base. Since evaluating every existing classifier is not feasible, we used the mentioned ones as a first selection. As a ranking system, we use the time needed for classifying a test fold (7588 instances) as well as creating a model based on a training fold (68296 instances). Furthermore, we look at the size of the resulting model. Note that all measurements found in Table 2 are median values over all folds of a 10 fold cross validation.

For creating the internal model structure, J48 is far superior compared to JRip, Ordinal and libSVM. The latter one even takes several days to compute, due to its complex mathematical nature. In this category, JRip with 14.5 hours places behind Ordinal with 7.1 hours, while J48 finishes in less than one hour.

However, since generating the model can be done beforehand, the classification steps needs to be efficient in order to produce fast results. Although there exist various optimization for classifying instances using support vectors, libSVM clearly under-performs, as it takes more than 9 hours to test approximately 7600 instances. J48 and JRip only need 23 and 32 seconds respectively for the same amount of instances, while the ordinal classifiers takes the lead with 19 seconds.

With those three as the only feasible classifiers left, we have a look at the size of their models. In this scenario, JRip clearly takes the win with only 128 rules compared to 5308 leaves of the decision tree learner J48. Surprisingly, Ordinal's tree only has 1132 leaves despite using J48 as a base classifier, while also working with an extra class label. This may explain its long creation time, as the tree is apparently highly pruned.

Ultimately, it comes down to the accuracy the classifiers can achieve, which is shown in Figure 38 and presents a clear win for both J48 and the ordinal classifier. Although the latter uses additional information extracted from the partially ordered sound levels, the overall prediction accuracy is slightly worse. Considering the fact that the resulting tree is almost five times smaller, this is yet a very good result.

A bit surprising is the bad result for the SVM classifier, only achieving approximately 57% accuracy. The huge amount of support vectors (62171) indicates that our particular dataset is not really suitable for this kind of learning scheme.

	Time for		Size (train fold)
	training (train fold)	classification (test fold)	
J48	0.8 h	23 sec	5308 leaves
Ordinal	7.1 h	19 sec	1132 leaves
JRip	14.5 h	32 sec	128 rules
libSVM	70.4 h	9 **h**	62171 support vectors

Table 2: Median values (over all respective train or test folds) of measurements for different classifiers.

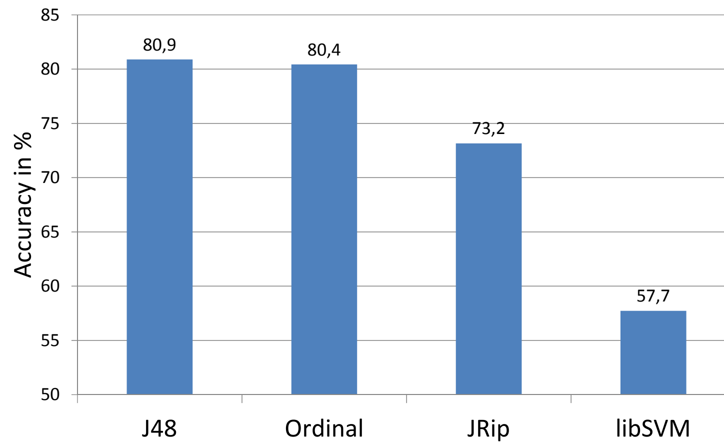


Figure 38: Accuracy of given classifiers on optimal feature composition.

This is also a factor we need to consider when we have a look at JRip's result. Recall that we use an optimal composition of our features, which is the same for all classifiers, but was evaluated using J48 only, introducing a bias towards it. If we evaluated our features using JRip, the results may differ. However, as outlined before, the far superior model creation time of J48 makes it more practical during feature evaluation.

5.2 Evaluation summary

After carefully examining different feature composition and sound level distributions as well as various classifiers, we can conclude that each approach has its own benefits and disadvantages. Consequently, the chosen features and classifiers are highly depended on the application's demands.

For example, as outlined before, some features may not be available for specific scenarios, hence they cannot be used, which leads to reduced accuracy. However, we also showed that one does not need all features to achieve a high accuracy and thereby may also willingly sacrifices a feature in order to produce a smaller model.

Small models can also be achieved by using different classifiers, as shown in the previous chapter. Although those may lack in performance and accuracy, they are more suitable for application or platforms, where memory is sparse. A good example of this would include the ordinal classifier we tested. Despite taking almost nine times longer to create a decision tree, the model itself is approximately five times smaller, while achieving virtually the same prediction accuracy.

As a conclusion, we will have another look at the approach which placed first in terms of accuracy, using all features with an equal-width approach and J48 as classifier. Table 3 shows common evaluation measurements of this composition. Considering the fact that we rely on user-generated data, which may contain compromised and contradictory measurements, an accuracy of almost 81% is a very good result. To the best of our knowledge, there exist no system, which can predict noise levels with an accuracy that high and is based on real-world data at once.

While we already mentioned accuracy as a guideline on how well a classifier is performing, recall or precision may be more suitable for some applications. For example, if it is more important to distinguish, whether an instance has one specific class label rather than belonging to one of the other classes.

The harmonic mean between recall and precision is represented by the F-Measure, while the mean absolute error is an indication on how many classification errors occurred. However, as we do not distinguish between severe and minor – predicting an adjacent class – errors, we will have an additional look at the confusion matrix shown in Table 4.

Accuracy	80.9%
Precision	80.8%
Recall	80.9%
F-Measure	80.8%
Mean abs. error	0.077

Table 3: Classification results using all features with J48.

		classified as (in dB)					
		<40	40-50	50-60	60-70	70-80	>80
is (in dB)	<40	82.4	11.1	2.7	2.6	1.0	0.2
	40-50	16.2	52.0	21.8	5.3	2.5	2.2
	50-60	1.7	6.6	65.0	23.8	1.7	1.3
	60-70	0.3	0.5	5.5	85.3	8.2	0.3
	70-80	0.2	0.5	0.7	13.8	82.6	2.2
	>80	0.2	1.0	0.7	1.7	11.6	84.7

Table 4: Confusion Matrix relative per class in % using all features with J48.

The confusion matrix shows that most instances are predicted correct with respect to their class label (the sound level). If the sound level was predicted wrong, an adjacent level was estimated in most case, minimizing severe prediction errors, such as labeling an instance as < 40dB, whereas the real value is in between 60-70dB. Only 0.3% were predicted wrong for this specific case, but 85.3% of all instance having a sound level between 60dB and 70dB were predicted correctly.

This shows that our model rarely makes severe prediction errors and tends to predict adjacent noise levels in case of a wrong classification. This robustness helps applications as they can rely on a more consistent result.

We have already mentioned some possible application scenarios in which *LOCAL* and especially the generated models can be of great benefit. Our initial motivation was to construct cheap yet accurate noise maps by using omnipresent smartphone sensors (see Chapter 1). So far we managed to gather various secondary features to successfully improve the accuracy of our approach. This resulted in a model – our knowledge – predicting sound levels on a categorical scale using information about the vicinity where the noise measurement was taken.

Using this model, we can now estimate noise levels for places we have not yet measured as long as the assumptions made by our model hold true. As predicting a noise level is also influenced by climatic terms – if we use the `Weather` feature – the model could fail to predict correct levels if we were to test it in a region differing strongly from the clime in Germany. This also applies to cultural differences, which may influence nearby building types or streets. Considering this, extending the usage of generated models past Europe is not recommended.

MINI

In Chapter 5.1.3, we discussed possible advantages of an equal-frequency approach when distributing the sound levels despite lacking in prediction accuracy. As opposed to the more static equal-width method, the former is more dynamic allowing to emphasize changes in noise levels, which are injected by altering the feature values, like time or number of nearby buildings.

Making use of this characteristic, the application *MINI*[46] combines noise maps with interactivity. It allows the user to change the time of the day and add buildings as well as remove them in real-time. Although still in development, *MINI* already shows interesting results, allowing to modify an existing area with regard to its noise level. This can provide insight into possible solutions against noise pollution. For example, removing certain buildings or changing the street surface in a particular area may decrease the noise level in the vicinity. Figure 39 presents an early version of *MINI*.

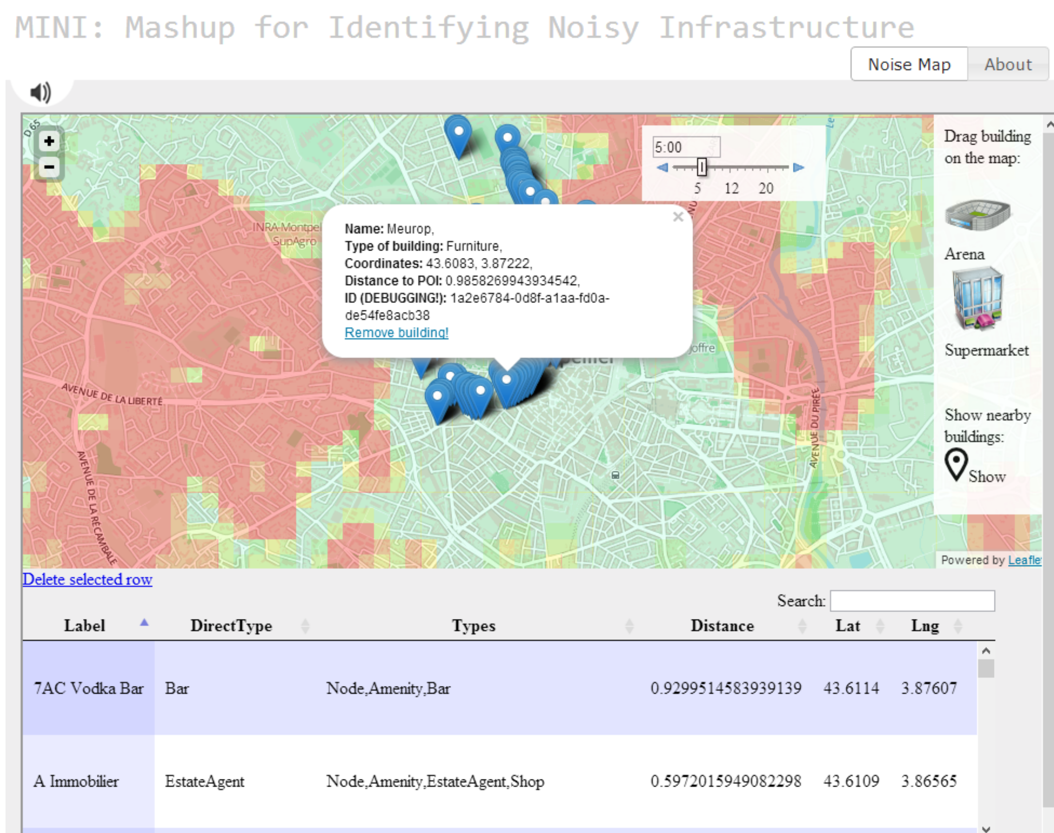


Figure 39: Early version of *MINI* showing a noise map for Montpellier. The blue pins resemble LGD entities found nearby. Using the icons on the right bar, the user can additionally place buildings on the map.

One way to visualize a noise map is to use a heatmap overlay. Each sound level corresponds to a color values, where green resembles quiet areas contrary to red areas, where the noise level would be higher. The gradient could also be adjusted dynamically to meet the application's needs. To construct such an overlay, we[47] mapped a grid onto the area we

[46] Mashup for Identifying Noisy Infrastructure: http://mini.ke.tu-darmstadt.de/MINI/ [accessed on 15.05.13]

[47] the MINI team: Axel Schulz, Jakob Karolus, Felix Mayer, Frederik Janssen, Heiko Paulheim, Immanuel Schweizer

want to display. As *MINI* is our contribution to the *AI Mashup Challenge 2013*[48] held in Montpellier, we used the French city for the prototype. To be able to respond in real-time, we constructed a database beforehand containing all necessary information about buildings and streets for each point of the grid. By doing so, we can load an initial noise map in mere seconds and modify it according to the user's input by manipulating the cached feature values.

[48] http://aimashup.org/aimashup13/doku.php [accessed on 15.05.13]

7 Conclusion and future work

In this thesis, we present a method to use participatory sensing and machine learning techniques as a mean to create cheap yet accurate noise maps of an area. We utilize auxiliary data sources, like nearby buildings or streets, to improve our knowledge of the vicinity. This helps us to make an elaborate decision about the noise level. After various optimizations, we are able to predict a sound level within a 10dB range for a given location with almost 81% accuracy. This shows that our approach is an adequate alternative for constructing noise maps as shown in *MINI* (see chapter 6). At the time of this writing, there exist only two noise mapping systems trying to reconstruct or predict noise levels for sparsely covered areas. As outlined in Chapter 3.1, *Ear-Phone* [34, 33] uses compressive sensing to reconstruct audio data and thus does not provide any conclusive solution for areas with no noise measurements. The work of Kaiser et al. [22] provides a first impression of the concept of using machine learning in conjunction with noise mapping. However, their approach does not include an evaluation on real-world data, in comparison to our work.

We also had a look at different feature composition as well as sound level distributions (see Chapters 5.1.2 and 5.1.3) and showed that each has its own benefits. Although smaller models lack in accuracy, they may be more feasible for some applications.

As we expect other data sources to make their debut in *LOCAL*, we designed a highly modular and extensible pipeline to cover the specific needs of many application scenarios. Thereby, we enable other developers to adapt or add data sources, as well as modifying feature modeling. Furthermore, users are able to construct a model with their very own feature composition if the need arises to omit one of the previously presented features.

As mentioned in the previous passage, we designed *LOCAL* to be highly customizable. Yet, due to limited time, we were only able to provide eight features using four different data sources. A major extension point would hence be the addition of entirely new data sources as well as optimize existing ones. This could include the weather source, as it is limited to Germany. For extending our approach to a world-wide basis, there certainly exists the need of a source which covers a broader area and provides a more elaborated API. Additionally, the OSM adapter could be modified by adding a daily live-sync compared to the static adapter at the moment. Such extensions would improve the data quality itself, which results in a better description of the POI. The same aspects also apply for all presented features, as there may be better ways to describe them, e.g. extending the query distance for nearby buildings, based on further evaluation or due to new data sources.

Fine-tuning of classifier parameters can also be considered for future work. As we omitted this due to lack of time, a through evaluation in this area could yield an increase in overall accuracy for all constructed models. Beside refining parameters of classifiers, the usage of other classification schemes could benefit the application as well. We only evaluated common algorithms, whereas other classifiers might be more suitable for this kind of task.

We already noted the possibility of a live-sync adapter for auxiliary data sources. This can also be realized for our initial source, as *Noisemap* continuously delivers new data, which could be used as training samples. This would ensure up-to-date noise measurements which are consistent with their surroundings.

In addition, the calculated sound levels from a live-sync model could be used for a heatmap implementation in an Android app or *Noisemap* itself. This would provide real-time noise prediction on the smartphone and can be used for the likes of augmented reality, e.g. warning the user when he enters a noisy region and suggesting alternative routes.

References

[1] Sören Auer, Jens Lehmann, and Sebastian Hellmann. Linkedgeodata – adding a spatial dimension to the web of data. *The Semantic WebISWC 2009*, 2009.

[2] M. Balmer, K. Meister, M. Rieser, K. Nagel, and Kay W. Axhausen. Agent-based simulation of travel demand: structure and computational performance of matsim-t. *Arbeitsbericht Verkehrs- und Raumplanung 504*, 2008.

[3] Mark Bilandzic, Michael Banholzer, Deyan Peev, Vesko Georgiev, Florence Balagtas-Fernandez, and Alexander De Luca. Laermometer: a mobile noise mapping application. In *Proceedings of the 5th Nordic conference on Human-computer interaction*, 2008.

[4] Max Braun, Ansgar Scherp, and Steffen Staab. Collaborative creation of semantic points of interest as linked data on the mobile phone. In *Semantic Web Challenge, ISWC 2009*, 2009.

[5] Jeffrey A. Burke, D. Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B. Srivastava. Participatory sensing. In *World Sensor Web Workshop at SenSys 2006*. ACM, 2006.

[6] World Wide Web Consortium. Resource description framework. `http://www.w3.org/RDF/`. [Online, accessed 02-May-2013].

[7] World Wide Web Consortium. Semantic web. `http://www.w3.org/standards/semanticweb/`. [Online, accessed 05-May-2013].

[8] World Wide Web Consortium. Sparql query language for rdf. `http://www.w3.org/TR/rdf-sparql-query/`. [Online, accessed 02-May-2013].

[9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[10] Ellie D'Hondt and Matthias Stevens. Community memories for sustainable urban living, 2009. BrusSense business plan, granted as a Prospective Research in Brussels IWOIB post-doctoral project.

[11] David L. Donoho. Compressed sensing. *IEEE Trans. Inform. Theory*, 52, 2006.

[12] Yasser EL-Manzalawy and Vasant Honavar. WLSVM: Integrating libsvm into weka environment, 2005. Software available at `http://www.cs.iastate.edu/~yasser/wlsvm`.

[13] H. Fletcher and W. A. Munson. Loudness of a complex tone, its definition, measurement and calculation. *The Journal of the Acoustical Society of America*, 5(1), 1933.

[14] Eibe Frank and Mark Hall. A simple approach to ordinal classification. In *12th European Conference on Machine Learning*, pages 145–156. Springer, 2001.

[15] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, 13(3), 1992.

[16] Marcus Goetz and Alexander Zipf. Openstreetmap in 3d – detailed insights on the current situation in germany. In *Proceedings of the AGILE 2012 International Conference on Geographic Information Science*, 2012.

[17] Michael F. Goodchild. Citizens as Sensors: the World of Volunteered Geography. *Geojournal*, 69:211–221, 2007.

[18] Julian Hagenauer and Marco Helbich. Mining urban land-use patterns from volunteered geographic information by means of genetic algorithms and artificial neural networks. *International Journal of Geographical Information Science*, 26:1–20, 2011.

[19] Mordechai Haklay. How good is volunteered geographical information? A comparative study of OpenStreetMap and Ordnance Survey datasets. *Environment and Planning B: Planning and Design*, 37, 2010.

[20] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1), 2009.

[21] Thomas Hofmann, Bernhard Schölkopf, and Alexander J. Smola. Kernel methods in machine learning. *Annals of Statistics*, 36(3), 2008.

[22] Christian Kaiser and Alexei Pozdnoukhov. Enabling real-time city sensing with kernel stream oracles and mapreduce, 2011. The First Workshop on Pervasive Urban Applications.

[23] Eiman Kanjo. Noisespy: A real-time mobile phone platform for urban noise monitoring and mapping. *Journal of Mobile Networks and Applications*, 15(4), 2010.

[24] Nicolas Maisonneuve, Matthias Stevens, Maria E. Niessen, Peter Hanappe, and Luc Steels. Citizen noise pollution monitoring. In *Proceedings of the 10th Annual International Conference on Digital Government Research: Social Networks: Making Connections between Citizens, Data and Government*. Digital Government Society of North America, 2009.

[25] Nicolas Maisonneuve, Matthias Stevens, and Bartek Ochab. Participatory noise pollution monitoring using mobile phones. *Information Polity - Government 2.0: Making Connections between citizens*, 15(1,2), 2010.

[26] Irene Garcia Martí, Luis E. Rodríguez, Mauricia Benedito, Sergi Trilles, Arturo Beltrán, Laura Díaz, and Joaquín Huerta. Mobile application for noise pollution monitoring through gamification techniques. In *ICEC*, Lecture Notes in Computer Science. Springer Berlin/Heidelberg, 2012.

[27] Pascal Neis, Dennis Zielstra, and Alexander Zipf. The street network evolution of crowdsourced maps: Openstreetmap in germany 2007–2011. *Future Internet*, 4(1):1–21, 2011.

[28] German Federal Ministry of Justice. Verkehrslärmschutzverordnung vom 12. juni 1990 (bgbl. i s. 1036), die durch artikel 3 des gesetzes vom 19. september 2006 (bgbl. i s. 2146) geändert worden ist. *Bundes-Immissionsschutzverordnung*, 2006.

[29] German Federal Ministry of Justice. Vierunddreißigste verordnung zur durchführung des bundes-immissionsschutzgesetzes (verordnung über die lärmkartierung) vom 6. märz 2006 (bgbl. i s. 516). *Bundes-Immissionsschutzverordnung*, 2006.

[30] OpenStreetMap. Osm statistics. `http://www.openstreetmap.org/stats/data_stats.html`, 2013. [Online, accessed 10-March-2013].

[31] The European Parliament. Directive 2002/49/ec of the european parliament and of the council of 25 june 2002 relating to the assessment and management of environmental noise. *Official Journal of the European Communities*, L189:12–25, 2002.

[32] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[33] Rajib Kumar Rana, Chun Tung Chou, Salil S. Kanhere, Nirupama Bulusu, and Wen Hu. Ear-phone-assessment of noise pollution with mobile phones. In *ACM SenSys*, pages 395–396, 2009.

[34] Rajib Kumar Rana, Chun Tung Chou, Salil S. Kanhere, Nirupama Bulusu, and Wen Hu. Ear-phone: an end-to-end participatory urban noise mapping system. In *ACM/IEEE IPSN*, 2010.

[35] Axel Schulz, Jakob Karolus, Heiko Paulheim, Max Mühlhäuser, and Immanuel Schweizer. Accurate pollutant modeling and mapping: Applying machine learning to participatory sensing and urban topology data. In *11th ACM Conference on Embedded Networked Sensor Systems*. ACM, 2013. In Submission.

[36] Immanuel Schweizer, Roman Bärtl, Axel Schulz, Florian Probst, and Max Mühlhäuser. Noisemap - real-time participatory noise maps. In *Second International Workshop on Sensing Applications on Mobile Phones*, 2011.

[37] Immanuel Schweizer, Christian Meurisch, Julien Gedeon, Roman Bärtl, and Max Mühlhäuser. Noisemap: multi-tier incentive mechanisms for participative urban sensing. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*. ACM, 2012.

[38] Eberhard Sengpiel. Table of sound pressure levels. `http://www.sengpielaudio.com/TableOfSoundPressureLevels.htm`. [Online, accessed 10-April-2013].

[39] Claus Stadler, Jens Lehmann, Konrad Höffner, and Sören Auer. Linkedgeodata : A core for a web of spatial open data. *Semantic Web Journal*, 2012.

[40] European Union. Future noise policy. com (96) 540 final. *European Commission Green Paper*, 1996.

[41] W3Schools. Rdf tutorial. http://www.w3schools.com/rdf/default.asp. [Online, accessed 02-May-2013].

[42] Die Welt. Online-atlas openstreetmap hat über eine million mitglieder. http://www.welt.de/newsticker/dpa_nt/infoline_nt/computer_nt/article112802104/Online-Atlas-OpenStreetMap-hat-ueber-eine-Million-Mitglieder.html, 2013. [Online, accessed 04-April-2013].

[43] Deutscher Wetterdienst. About dwd. http://www.dwd.de/bvbw/appmanager/bvbw/dwdwwwDesktop?_nfpb=true&_pageLabel=_dwdwww_wir_ueberuns_kurzportraet&activePage=&_nfls=false. [Online, accessed 04-April-2013].

[44] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 3 edition, 2011.

[45] Moustafa Youssef and Ashok Agrawala. The horus wlan location determination system. In *Proceedings of the 3rd international conference on Mobile systems, applications, and services*, MobiSys '05. ACM, 2005.

[46] Dennis Zielstra and Alexander Zipf. A comparative study of proprietary geodata and volunteered geographic information for germany. In *13th AGILE International Conference on Geographic Information Science*. ACM, 2010.