UNIVERSITY OF BERN

BACHELOR THESIS

Machine Learning for Indoor Positioning

Author:
Joel NIKLAUS

Supervisors:
Prof. Dr. Torsten BRAUN
Dr. Zhongliang ZHAO
Jose Luis CARRERA

A thesis submitted in fulfillment of the requirements
for the degree of Bachelor of Science

in the

Communication and Distributed Systems Research Group
Institute of Computer Science

November 14, 2017

Declaration of Authorship

I, Joel NIKLAUS, declare that this thesis titled, "Machine Learning for Indoor Positioning" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"In a world that's changing really quickly, the only strategy that is guaranteed to fail is not taking risks."

Mark Zuckerberg

University of Bern

Abstract

Faculty of Science
Institute of Computer Science

Bachelor of Science

Machine Learning for Indoor Positioning

by Joel NIKLAUS

Nowadays, smartphones can collect huge amounts of data in their surrounding environment with the help of highly accurate sensors. Since the combination of the Received Signal Strengths of surrounding Access Points and sensor data is assumed to be unique in every location, it should be possible to use this information to accurately predict a smartphone's location. As it is very difficult to derive the correlation between these values, we must use machine learning methods. As part of this project, we have developed an Android application that is able to distinguish between rooms on a floor and special landmarks within a room. This has been accomplished using machine learning methods based on the Java library Weka. Ultimately, we hope to include this application into an indoor tracking system in order to improve its accuracy.

Acknowledgements

On this page I want to thank every person who helped me in any way in completing this thesis. In particular I want to give sincere thanks to my supervisors Prof. Dr. Torsten BRAUN, Dr. Zhongliang ZHAO and Jose Luis CARRERA for their help and support during my project. Furthermore, I yield Melissa CHANG and Catriona REID special thanks for proof reading my thesis and helping me greatly with the English language. In addition I want to thank my flatmates in Exeter and Bern warmly for their patience while I was disturbing them by conducting my experiments in the living room. Last but not least many thanks to my parents, siblings and friends for their emotional support.

Contents

List of Figures

List of Tables

List of Abbreviations

ML	**M**achine **L**earning
IL	**I**ndoor **L**ocalization
IP	**I**ndoor **P**ositioning
ITS	**I**ndoor **T**racking **S**ystem
RSS	**R**eceived **S**ignal **S**trength
Indoloc	This **Indo**or **Loc**alization System
KNN	**K** **N**earest **N**eighbour

This thesis is dedicated to everyone who helped me on the project, except for that guy who yelled at me in Migros when I was seven because he thought I was being "too rowdy".

You're an asshole, sir.

Chapter 1

Introduction to the Thesis Topic

In this chapter we are giving a short introduction to the topic and the motivation. The goal of this project is to estimate the indoor locations of smart phone users on the room level accuracy using ML methods.

1.1 Indoor Localization

High localization accuracy within buildings would be very useful - in particular, large complex buildings like shopping malls, airports and hospitals would be well served by this feature. It would make orientation within these highly complicated structures much easier and would diminish the need for big plans scattered all around these buildings.

However, walls, roofs, windows and doors of the buildings we live in greatly reduce the GPS signals carried by radio waves because it operates on a relatively high frequency of 1575.42 MHz (L1 signal) and 1227.6 MHz (L2 signal). This results in a severe loss of accuracy in GPS data inside buildings. (MiTAC, 2017)

Different solutions already exist for indoor localization of mobile devices such as Pedestrian Dead Reckoning (PDR) and WiFi fingerprinting based methods. In PDR at every step/current location of the user his/her direction and therefore future location is predicted using inertial sensors. In WiFi fingerprinting, the Received Signal Strength (hereafter referred to as RSS) values of several access points in range are collected and stored together with the coordinates of the location. A new set of RSS values is then compared with the stored fingerprints and the location of the closest match is returned. (Xiao et al., 2016)

1.2 WiFi and Sensor Data

In contrast to outdoors, building interiors normally have a large number of different WiFi access points constantly emitting signals. So why do we not use these to predict the user's location? By scanning the area around the device, we can measure the received signal strength of each of the nearby access points. And because there typically are so many of them, we presume that the list of all these values combined is unique at every distinct point in the building.

Furthermore, we can strongly assume that these values are also constant over time as the access points are fixed in place and are constantly emitting signals of the same strength. Of course, there may be occasional changes, for instance if the network is remodelled, but we expect these changes to be infrequent.

In addition to the RSS values, we also suggest using the earth's magnetic and gravity field and collecting other data using the sensors available in modern smart phones.

1.3 Machine Learning

In this way, we can collect lots of labelled location data of the building. However, because each data point may contain a very large number of WiFi access point RSS values and magnetic field values, the data is very complex. Therefore, we propose using supervised Machine Learning (herafter referred to as ML) methods to make sense of this large amount of collected data. By training a classifier (supervised learning algorithm such as K-Nearest-Neighbour) on the collected labelled data, rules can be extracted. Feeding in the actual live data (RSS values, magnetic field values, etc.) of a moving user, the trained classifier can then predict the user's location. We propose using machine learning to solve this task because the data is highly complex, containing many different features, such as RSS values, magnetic field values and other sensor data. We expect the supervised learning algorithms to discover patterns in the data which can then be used to differentiate between different rooms for instance. (Mascharka and Manley, 2015; Mascharka and Manley, 2016)

Chapter 2

Background Information and Theory

This chapter gives necessary background information for the topics covered in this thesis.

2.1 Motivation for Using Machine Learning to Improve the Indoor Tracking System

The Indoor Tracking System, hereafter referred to as ITS, (Carrera, Zhao, Braun, Li, and Neto, 2016; Carrera, Zhao, Braun, and Li, 2016; Li et al., 2016; Shala and Rodriguez, 2011) can continuously predict the user's location based on the location of the user a short time ago and the orientation and movement of the device. Basically, it suggests a collection of possible points the user could be at. It can predict the user's location with an accuracy of 1.7 meters.

The aim of the indoor localization system presented in this thesis (hereafter referred to as Indoloc) is to improve the accuracy of the ITS using ML. We hope that this can be done by excluding all the possible locations of the user of one room if the Indoloc predicts the other and by using landmarks. A landmark is defined as a small area within a room. Thus, when a landmark is recognized by Indoloc, the ITS can then tell with high confidence that it is located in a certain very small area. This makes the ITS more precise. (Deng et al., 2016; Wang et al., 2016)

Figure 2.1 shows how Indoloc can improve the ITS. The red points symbolize the collection of possible locations the user could be at, suggested by the ITS. The five boxes represent the landmarks. These landmarks are small imagined areas inside a room. Indoloc predicts the landmark "top left" with the green background. As a consequence, the ITS can exclude the points which are far away, namely the ones crossed out.

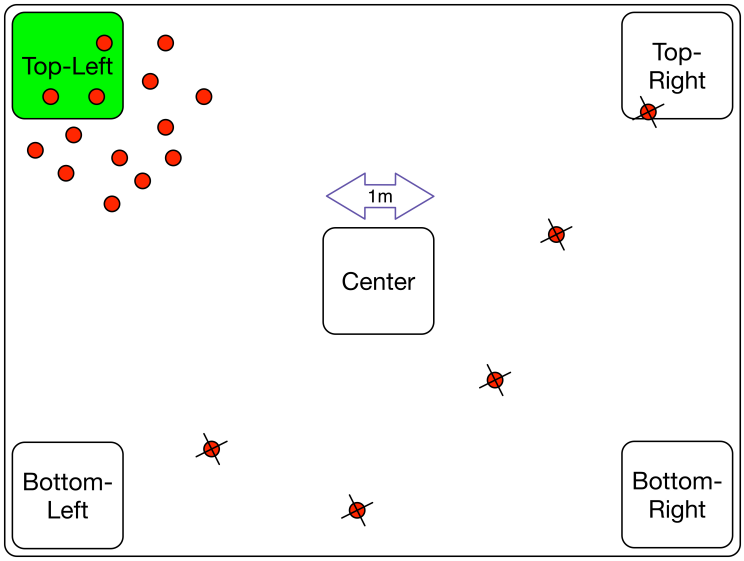

FIGURE 2.1: Five landmarks and the collection of red points predicted by the ITS.

2.2 Reasons for Using Machine Learning

Machine Learning is very suitable for analyzing big amounts of data. The supervised learning methods that are used in this work read the given training data and then build a model of it, which tries to distinguish the given classes. This model can then be applied to the data points collected by the user during the experiment and, in this way, it can predict the class (room or landmark respectively) of the user. By using Machine Learning methods we can take advantage of the modern smart phone's ability to collect huge amounts of data about what is happening around it.

2.3 Machine Learning Workflow

In this section we are giving a short introduction about the workflow we used to achieve the best accuracy possible for the tested datasets. We used an article about a typical machine learning workflow (Google, 2017) as a guideline for our workflow.

1. **Data Preprocessing**
   First, the data has to be cleaned. Redundant or insensible information has to be deleted.

2. **Attribute Selection**
   Second, the right attributes have to be selected. Some attributes may not contribute any useful information to the model and can therefore be discarded. Specifically for this thesis it is discussed in Section 4.1 and 4.3.

3. **Feature Engineering**
   In this part of the workflow, we try to create new features, for instance in Section 2.5.2, or modify existing features to improve the accuracy (Section 4.4).

4. **Find Base Learners**
   In this section, we find the best base ML method.

5. **Find Meta Learners**
   In this step, we improve the performance using meta learners. A meta learner is defined as a combination of one or several different base learners (conventional ML algorithm). This is described in more detail in Section 4.6.

6. **Hyper-Parameter search**
   Finally, we use methods like autoweka, gridsearch, multisearch or trial-and-error to tweak the parameters of the chosen ML methods in order to further improve the accuracy (Section 4.5).

This workflow is an iterative process. In order to achieve the best results it may have to be repeated several times.

2.4 Chosen Machine Learning Algorithms

As Weka provides a very big amount of efficiently implemented machine learning algorithms we could test lots of them without spending efforts of implementing them from scratch. An overview can be found on `http://wiki.pentaho.com/display/DATAMINING/Classifiers`. The following list includes the algorithms that have been implied in this thesis. Detailed information about the algorithms can be found in the method `addClassifiers()` in the class `https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/AbstractTest.java`.

- Bayes
  - NaiveBayes (see John and Langley, 1995)
- Functions
  - LibSVM (Library for Support Vector Machines, see Chang and Lin, 2017)
  - Logistic (see Cessie and Houwelingen, 1992)
  - MultilayerPerceptron (Neural Network)
  - SMO (Sequential Minimal Optimisation) (see Platt, 1998; Keerthi et al., 2001; Trevor Hastie and Robert Tibshirani, 1998)
- Trees
  - J48 (see Quinlan, 1993)
  - RandomForest (see Breiman, 2001)
- Lazy (Instance Based)
  - IBk (Implementation of the K-Nearest-Neighbour Algorithm, see Aha and Kibler, 1991)
  - KStar (see Cleary and Trigg, 1995)
  - LWL (Locally Weighted Learning, see Frank, Hall, and Pfahringer, 2003; Atkeson, Moore, and Schaal, 1996)
- Meta
  - AdaBoostM1 (Adaptive Boosting, see Freund and Schapire, 1996)

- Bagging (see Breiman, 1996)
- Dagging (see Ting and Witten, 1997)
- Decorate (see Melville and Mooney, 2003; Melville and Mooney, 2004)
- Grading (see Seewald and Fuernkranz, 2001)
- LogitBoost (see Friedman, T. Hastie, and R. Tibshirani, 1998)
- RandomSubSpace (see Ho, 1998)
- Stacking (see Wolpert, 1992)
- Vote (see Kuncheva, 2004; Kittler et al., 1998)

Because an explanation of all the tested algorithms would go beyond the constraints of this thesis we are giving the following recommendations for the interested reader: A good website to get an overview on `http://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/` and a good article for more details on `http://alex.smola.org/drafts/thebook.pdf`.

2.5 Features

In a Machine Learning project the attributes of the classes are denoted as features. Each feature is describing an aspect of the classes. In our case features are our measurements, for instance an RSS value. For the ML project to deliver a good prediction accuracy it is very important to select the right attributes/features and to also modify certain features or even create new features out of existing features. This is part of the ML workflow described in Section 2.3.

In the following sections both the features used in the system and the ones taken into consideration, but then found to be useless, are introduced. Each feature corresponds to one column in the dataset and would be referred to as an attribute in Weka.

2.5.1 Features used in the experiments

In the following sections, the features which are actually used in the running system are presented.

RSS

The RSS values provide the core data as they contribute the most to the performance of the ML methods. The smart phone scans the surrounding access points, obtains and registers the RSS values of each access point. These values depend on the distance to the access point as well as on the existence of obstacles, such as walls or furniture, between the access point and the device. Normally the RSS values in our datasets were between -20 and -90.

Magnetic Field

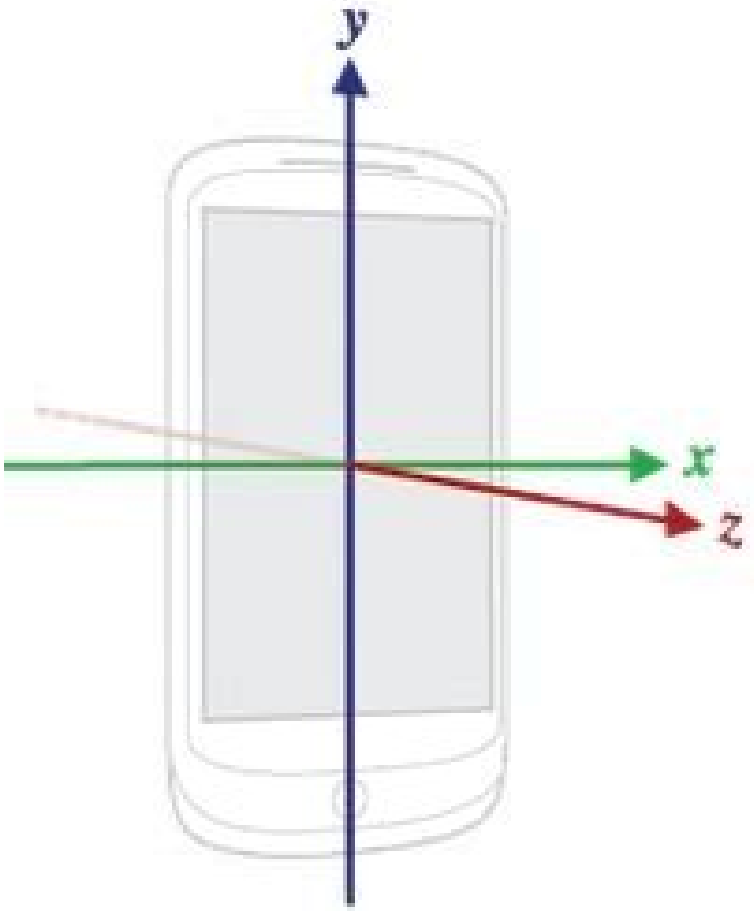

FIGURE 2.2: The values in the device's coordinate system.

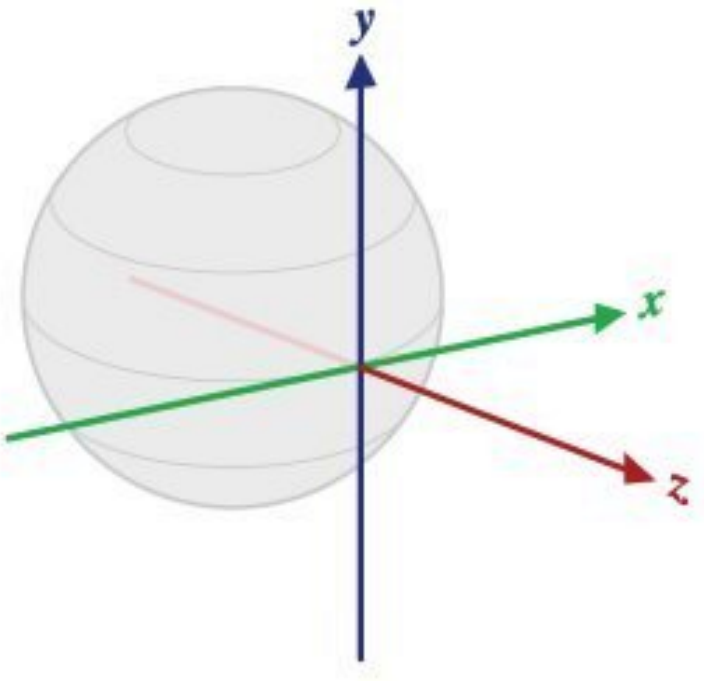

FIGURE 2.3: The values in the earth's coordinate system.

The device's sensors measure the magnetic field in the device's coordinate system. As the user walks around, the orientation of the device may change all the time. We would therefore have to collect all possible values from every orientation in every point for the training phase. This would result in a huge amount of data and the training performance would be extremely inaccurate.

In addition to the raw magnetic field values we can also derive data from the accelerometer measuring gravity in the device's coordinate system. By using the Android built in function `SensorManager.getRotationMatrix()` and providing the magnetic field and gravity values, we can get the rotation matrix `R` and the inclination matrix `I`. Using these two matrices, we have two methods making use of the raw data (coded in https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/main/java/ch/joelniklaus/indoloc/models/SensorData.java):

Magnetic Processed With the help of `R` we then can do a change of the basis from the device's to the earth's coordinate system to get the processed magnetic values `mp`. (see 2.1)

$$\begin{pmatrix} 0 \\ mp \\ mp \end{pmatrix} = R * m \tag{2.1}$$

Now we have got the magnetic field data at a specific point in the earth's coordinate system. The x value is in East-West Direction, the y value is in North-South Direction and the z value is perpendicular to the center of the earth. As the magnetic field of the earth only goes from one pole to the other the x value is always 0 and is therefore not used as a feature.

Gravity Magnitude and Geomagnetic Magintude In order to obtain the gravity magnitude `grm` (in z direction) we multiply the gravity vector `g` with `R`. (see 2.2)

$$\begin{pmatrix} 0 \\ 0 \\ grm \end{pmatrix} = R * \begin{pmatrix} g_1 \\ g_2 \\ g_3 \end{pmatrix} \tag{2.2}$$

To obtain the geomagnetic magnitude `gem` (in y direction) we multiply the magnetic vector `m` with $I * R$. (see 2.3)

$$\begin{pmatrix} 0 \\ gem \\ 0 \end{pmatrix} = I * R * \begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix} \tag{2.3}$$

Additional information In order to reduce extreme variations which might disturb the ML algorithms, we apply a low-pass-filter to the newly collected data. For the alpha value we chose 0.75. So the last value has a weight of 0.75 and the newly collected value a weight of 0.25. The accuracy of the magnetic field sensor in the devices we tested is 0.15 µT. But still the data the sensor outputs contains many more decimal places. In order to exclude the noise, we round the values to 1 µT. The values obtained by measuring the magnetic field provide an improvement to the overall accuracy.

Light

We also thought about including data collected from the light sensor because, for instance, a room facing a window will clearly be brighter than one surrounded by walls only. As can be seen in Section 4.3.3 this does improve the prediction accuracy, however, these assumptions are not stable over time. In the night, there may be no difference concerning light at all between the two previously mentioned rooms. Or if there is a cloudy day the overall light strength would probably be much less as compared to that on a sunny day. Thus, it might be appropriate to work with light differences to a representative data point. This data point would have to be chosen carefully and would have to be recorded first.

We include this suggestion here because it improves the result but it still has to be tested if it is worth integrating it into a system in a productive environment.

2.5.2 Non-useful features

In the following sections the features which have been taken into account but have been judged as not helpful are presented. These features are therefore not included into the running system, as they would only produce additional overhead and slow down the generation and evaluation of the ML models.

Ambient temperature

At first we thought about including the ambient temperature as another feature because there may be characteristic small temperature differences between rooms which could help in making predictions.

However, we estimate this feature to be rather unpredictable. For instance when someone turned the heating up in one specific room, the temperature would change drastically. As a consequence, we would suddenly receive incredibly conflicting

values. Or imagine someone opens the window on a very cold day. Even without any human interference, different seasons or poor isolation of walls or windows could have an influence on temperatures in the same room. Apart from that, there are still lots of devices which do not yet have any temperature sensors.

Relative Humidity

Another idea would have been to include relative humidity. In comparing a bathroom and an office, for instance, it would be a fair assumption that the relative humidity would be higher in the former than the latter. But, as with the ambient temperature, we think that it is too volatile overall. An open window on a rainy day or even just a hot steaming tea can change the humidity landscape of a room. And here as well, there are not many devices which already have a relative humidity sensor built in.

Pressure

Relative pressure differences are utilized in the ITS to distinguish between different floors of a building. But this work is only part of the ITS and only has to provide predictions on the same floor. Therefore, it does not make sense to include it in this system. Furthermore, the Motorola test device was not even equipped with a pressure sensor.

Mean and Variances of RSS

One idea was to include the mean and the variances of the RSS values. But as experiments have been able to show, this does not improve the overall accuracy. This is probably the case because it does not add additional information to the ML algorithms but just alters and adds pre-existing ones.

GPS

An idea is to include the latitude and longitude gained from the GPS to help prediction close to the windows. But as our tests in Section 4.3.3 have shown the accuracy is decreased if it is included.

2.6 Weka

Weka, standing for Waikato Environment for Knowledge Analysis, is an open source machine learning library programmed in Java developed by the University of Waikato, New Zealand, and can be found on `http://www.cs.waikato.ac.nz/~ml/weka/`.

Two reduced versions for Android [1] [2] have been considered and the first version (from the developer with user name rjmarsan) has been chosen. It uses Weka 3.7.3 and mainly does not include the GUI (Graphical User Interface) parts of the system which are not used in this application.

[1] rjmarsan: `https://github.com/rjmarsan/Weka-for-Android`
[2] Shookit: `https://github.com/Shookit/android-ml-weka`

2.6.1 Advantages

The main reason why we chose the library Weka, is the huge amount of efficiently implemented ML algorithms it provides. It offers a wide range of both supervised and unsupervised learning algorithms. Furthermore, in addition to the Java interface there is both a command line interface and a graphical user interface available.

2.6.2 Disadvantages

Although the introduction documentation [3] [4] provides you with some information to get started, the Javadoc [5] is not of much use. It only provides one very small sentence to most of the methods (functions). This sentence mostly does not add any additional details to the information from the method's name.

The Javadoc for the classes is slightly better though, providing some explanation how to use it. The design of the methods is very close to the use of the command line. So usually a string array of cryptic options has to be provided in order to configure the classifier. Sometimes it can be done using setter methods, but if so it is poorly documented what exactly is altered by setting a certain value.

There is a mailing list [6] and a forum on Pentaho [7] where questions concerning Weka can be asked. But unfortunately the active community does not seem to be very large. For my problems at least, we had great difficulties finding answers through these two channels.

2.7 Android App

2.7.1 Implementation

If the reader is interested in knowing how to implement an Android app we can recommend the following web resources: Android Developers on https://developer.android.com/index.html and Android Studio on https://developer.android.com/studio/index.html.

2.7.2 Reasons for Choosing the Android System

The most important reason which lead to the choice of Android as a (first) platform of the app was the ITS which is being implemented in Android. On top of that, Weka offers a Java interface which makes it very easy to integrate into an Android app.

2.7.3 Tested Mobile Phones

First, we tested the Android app on the Emulator in Android Studio. This was fine for examining how the user interface behaved but after we added the WiFi, Sensor and GPS collection, we had to test it on real phones. We used the following two Android phones: Nexus 4 (LG, Android Version 5.0, API, Level 22, Accurate Specifications on https://en.wikipedia.org/wiki/Nexus_4) and Moto X Style (Motorola, Android Version 6.0, API Level 24, Accurate Specifications on https://

[3] Programmatic Use: https://weka.wikispaces.com/Programmatic+Use
[4] Weka in Java Code: https://weka.wikispaces.com/Use+WEKA+in+your+Java+code
[5] Javadoc: http://weka.sourceforge.net/doc.stable/
[6] Mailing List: https://list.waikato.ac.nz/mailman/listinfo/wekalist
[7] Pentaho Forum: http://wiki.pentaho.com/display/DATAMINING/Pentaho+Data+Mining+Community+Documentation

www.digitec.ch/en/s1/product/motorola-moto-x-style-570-32gb-21mp-black-mobile-phones-5339851).

Chapter 3

Implementation and Experimentation

In this chapter, the experiments and their setup are explained. Further, we present the datasets which are used in the experiments.

3.1 General Remarks

The larger the physical gap between rooms or landmarks respectively, the larger are the differences in the measured features. As a consequence, the space between the different classes in the hyperspace is increased, which makes it easier for the ML algorithm to distinguish between the classes.

3.2 Implementation

This section gives some details about how to use Weka as open source ML library to make the predictions of rooms or landmarks respectively. Figure 3.1 shows the dataflow and the different components of the Android app in a visualized way. Sensor and RSS values are measured by the device and received in the Android app. This data is then passed to a Weka component, which trains the ML algorithms. The trained algorithms are then evaluated with test data to find the one with the highest prediction accuracy. Finally, the best trained ML algorithm is then used for live testing finally returning the device's location.

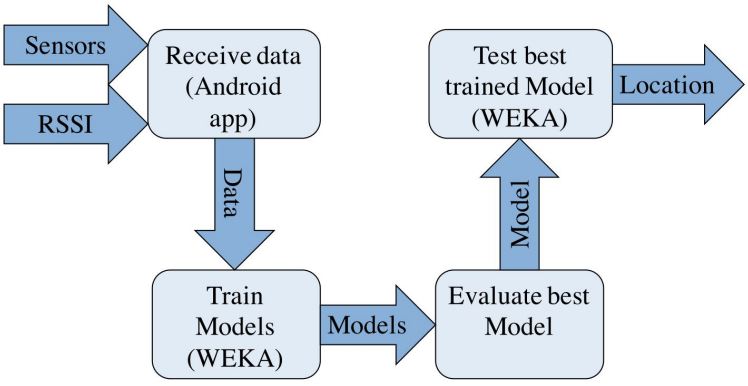

FIGURE 3.1: The architecture of the implemented android app.

3.3 Machine Learning Applications

3.3.1 Room Recognition

In the room recognition phase we want the system to distinguish several rooms on the same floor. As mentioned above, the sensor accuracies are not high enough to enable the ability to distinguish data points collected at the border of two rooms. Our best practice was leaving one square metre of space at the doors without any data points collected.

3.3.2 Landmark Recognition

In the landmark recognition phase we want the system to distinguish several landmarks inside the room. As explained before, a landmark is defined as a small area within a room. Typically a landmark was of one square metre size and between each of the landmarks we left at least 2 metres space. This was our best practice and there may be better ways to do this. So in a small room (around 3x3 metres) we would have two landmarks, so one at each end. In a normal office-sized room (around 5x5 metres) we would have four landmarks, one in each corner. In a big room (around 7x7 metres) we would have five landmarks, one in each corner and one in the centre. In our experimental environment there were no bigger rooms available. (Figure 3.2)

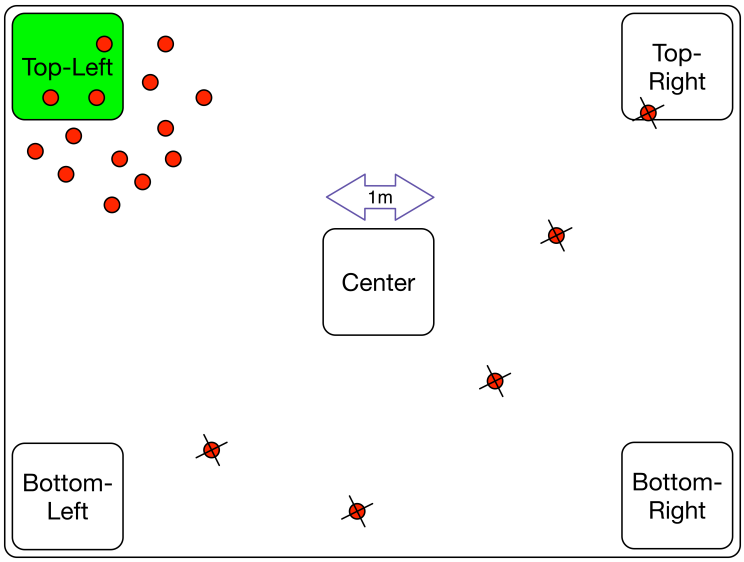

FIGURE 3.2: Five landmarks and the collection of red points predicted by the ITS.

3.4 Data Collection Methodology

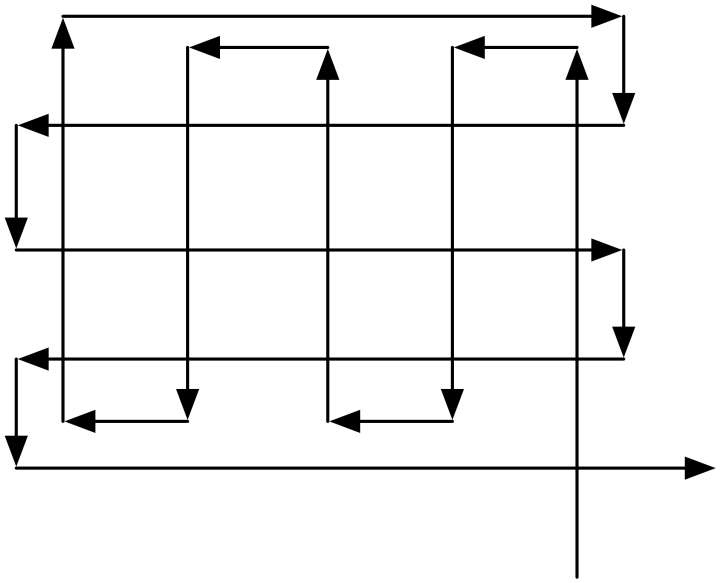

FIGURE 3.3: The grid pattern used to collect the data points.

The data collection for both the training set and the test set have been done in a grid pattern, as shown in Figure 3.3. So we started with the phone in one corner of the room or landmark respectively. Then, we continuously moved the phone back and forth in parallel vertical lines. Once we covered the entire area in vertical lines, we turned the device by 90 degrees and did the same in parallel horizontal lines until we arrived at the diagonally opposing corner. The distance between these lines was typically around 5cm. Using this grid pattern method, a very tightly woven net could be laid across the area. Therefore, the diversity of the collected data points could be maximized. Additionally, this way of collection data is very fast. For instance, it takes 20 minutes to finish the entire data collection in the third floor of institute building in Bern.

The training set and the testing set have been collected in two different gatherings within two hours on the same day. Because when they were collected in the same gathering and split up afterwards, Random Forest normally had over 99% testing set accuracy in offline testing on the computer. But in live testing on the smart phone this accuracy was nowhere close (< 70%).

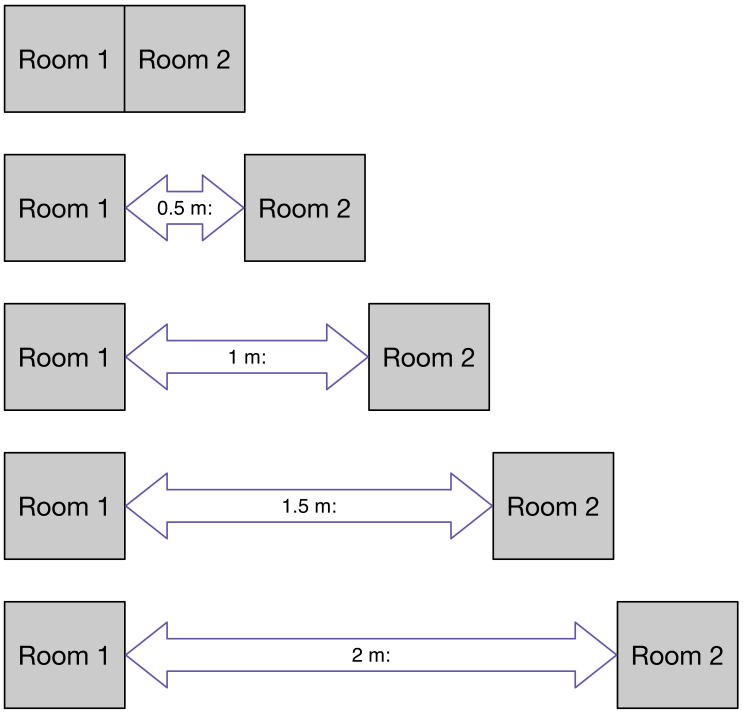

FIGURE 3.4: Distance between rooms.

In supervised ML projects a class denotes the prediction output of the given ML algorithm. In our room prediction case the classes would be the different rooms and in analogy in our landmark prediction case the classes would be the different landmarks.

Generally, the physically closer together the classes are, the lower is the accuracy of the algorithms. This is due to small differences in the measured values (as shown in Figure 3.4).

So on the one hand, when the distance between the classes is large, the measured values have greater differences between each other. This makes it easier to distinguish the classes and results in a higher prediction accuracy. However, this prediction is also less useful because there is a big uncategorized space in between the classes (as shown in the bottom row in Figure 3.4).

On the other hand, if the distance between the classes is small, the measured values have smaller differences between each other. This of course exacerbates the problem of distinguishing the classes as the data points close to the border but in different classes have very similar values. But this prediction is much more helpful since more of the space can be categorized into a class (as shown in the top row in Figure 3.4).

So we have to find an optimal solution in between the two above extremes, which gives us a good accuracy but also does not come with too much uncategorized space. Gathering from our experience we need an approximate distance of 1.5 metres between different classes to achieve an prediction accuracy of 85%.

In the method `registerListeners()` in the Sensor Helper class (on https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/main/java/ch/joelniklaus/indoloc/helpers/SensorHelper.java) we set the collection delay to `SENSOR_DELAY_NORMAL`. Looking at the Android Documentation (on https://developer.android.com/guide/topics/sensors/sensors_overview.

html) we read that this delay is set to 200000 microseconds. Therefore, the sensor returns at least (!) 5 values per second. So we collected around 5 data points per second walking at a constant rate.

3.5 Datasets

A dataset consists of two separately collected subsets, the training and the test set. The training set usually is larger than the test set. Normally we use splits like 70% training set and 30% test set or 80% and 20%.

3.5.1 Bern Dataset

The dataset considered for room recognition has been recorded on the third floor of the Computer Science building of the University of Bern at Neubrückstrasse 10, as shown in Figure 3.5. It can be found on the github repository on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room.

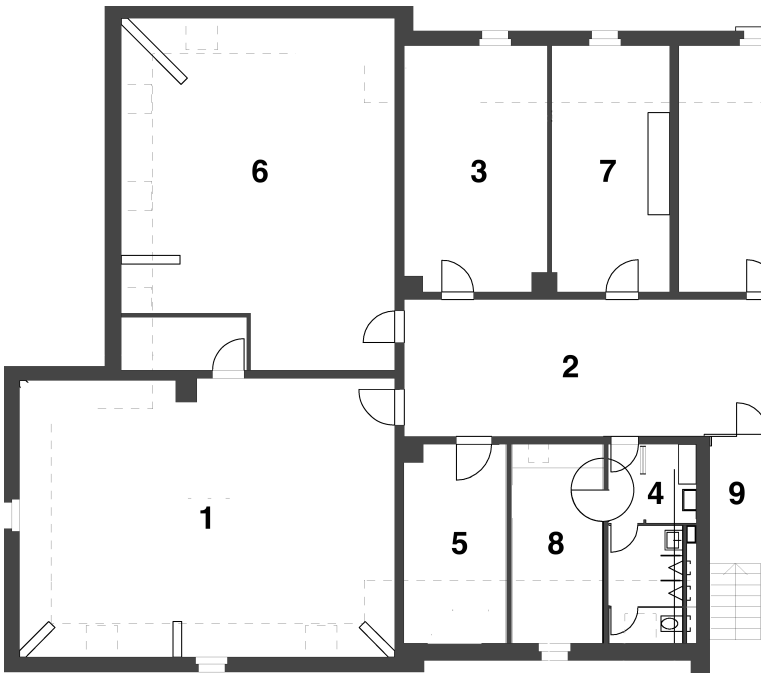

FIGURE 3.5: The testing environment in Bern.

The training set contains 14569 data points in total. 3061 data points were collected in the biggest room (1) and 514 in the smallest room (4). As mentioned in Section 3.4 around 5 data points can be collected per second. Therefore, collecting the whole training set required around 50 minutes.

The test set contains 8624 data points with 2164 in room 1 and 291 in room 4. Collecting the whole test set required around 30 minutes. In this dataset we collected all the features described above. So, in this section it is also described which of the features improve the accuracy and which of them could be left out.

This dataset is used to conduct experiments on room recognition level and on which features are beneficial for accuracy.

3.5.2 Exeter Dataset

The dataset considered for landmark recognition has been recorded on the first floor of the living room in Block C in James Owen Court on Sidwell Street in Exeter, an apartment complex owned by the University of Exeter, as shown in Figure 3.6. It can be found on the github repository on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/exeter/landmark.

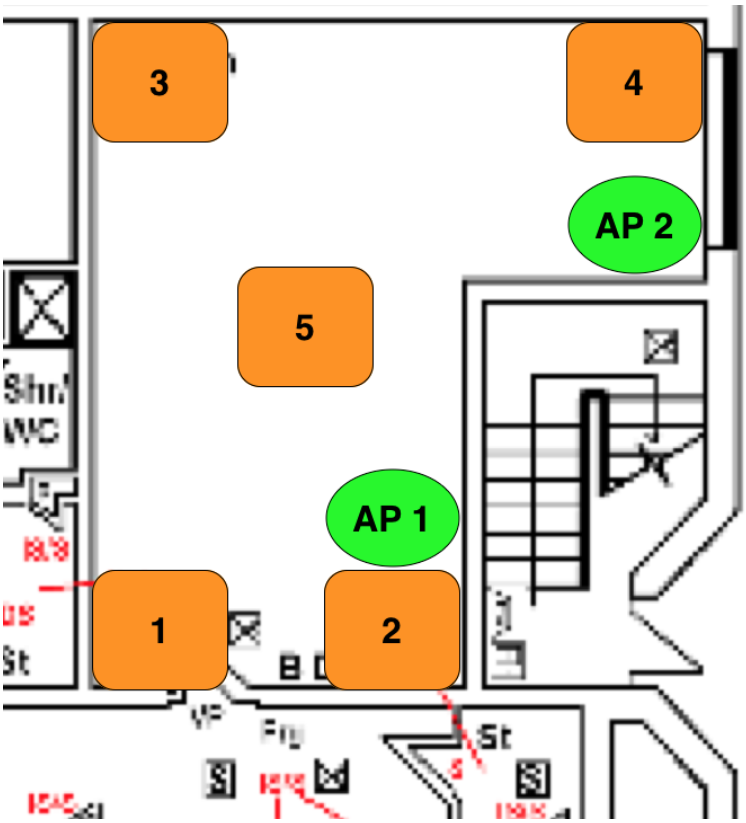

FIGURE 3.6: The testing environment in Exeter.

The training set contains 2522 data points in total. As each of the landmarks are of equal size we collected a little over 500 data points in each of them. Collecting the whole training set required approximately 10 minutes.

The test set consists of 1269 data points with a little over 250 data points in each landmark. Collecting the whole test set required approximately 5 minutes.

In this dataset we only collected the magnetic field values in the y and z directions, and the RSS values, because at the time of collection we presumed these to be important features. There are many different machine learning methods and they can each be parametrized in a variety of ways. Furthermore, each feature combination can be tested if it improves the result. So, in the following sections we am confining ourselves to the most expressive findings.

This dataset is used to conduct experiments on the landmark recognition level and to check if removing duplicates improves the result (4.2).

In every section there is a link available at the top which leads directly to the Java class doing the described experiment.

Chapter 4

Experimental Results

This chapter describes the results of the experiments. On the github repository (https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room) all the datasets we collected for the experiments are provided for the interested reader.

4.1 Attribute Selection

The information gain denotes the value a certain attribute has for the overall prediction. If an attribute with a high information gain is removed, we can strongly assume that the prediction accuracy will decrease more than when an attribute with a low information gain is removed. Finding out the information gain of each attribute lets us rank the attributes based on their usefulness.

In order to evaluate the information gain (see Frank, Hall, Holmes, et al., 2010) of each attribute on the dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room), we run the InfoGainAttributeEval (see http://weka.sourceforge.net/doc.dev/weka/attributeSelection/InfoGainAttributeEval.html) ranking algorithm in Weka, getting the following result in Table 4.1.

TABLE 4.1: The information gain of each attribute using the InfoGainAttributeEval ranking algorithm.

Attribute Name	Information Gain	Attribute Index
light	2.3540	3
latitude	2.3427	16
longitude	2.1380	17
rssValue5	1.9490	23
rssValue4	1.9462	22
rssValue3	1.9288	21
rssValue1	1.8435	19
rssValue2	1.8426	20
rssValue6	1.8079	24
rssValue0	1.5762	18
rssValue8	1.5582	26
rssValue9	1.4883	27
rssValue7	1.2531	25
geomagneticMagnitude	0.6001	13
magneticProcessedZ	0.3557	15
magneticProcessedY	0.3459	14
gravityMagnitude	0.0573	12
pressure	0	4
magneticZ	0	11
magneticY	0	10
relativeHumidity	0	5
gravityX	0	6
gravityY	0	7
gravityZ	0	8
magneticX	0	9
ambientTemperature	0	2

Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

This ranking provides information about the probable importance of the features. The top ranked feature contains the greatest information gain and is therefore probably very important for the predictions made by the classifiers used later on. The 9 features at the bottom of the ranking have an information gain of 0. Therefore, we already know that these 9 attributes are only cluttering the data and are of no use to us. So we can already delete these out of the dataset. This results in no difference in accuracy but in a decrease in both training and testing time as the algorithms have to consider less data.

4.2 Duplicates Removal

As described in Section 3.4 and depicted in Figure 3.3 there are some intersections in the path covered by the researcher to collect data. At these intersections or at locations very close to each other it is possible that every attribute of two rows in the dataset have the same values. These are denoted as duplicate data points.

The Duplicates Test (on https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/experiments/DuplicatesTest.java) checks if the accuracy is increased when all the duplicate data points are removed. After removing the duplicate data, it is clear that every data point is unique. A data point is a duplicate of another data point if all the correspondent feature values are exactly the same and they belong to the same class - simply put, if both data points are exactly the same.

TABLE 4.2: The accuracy change if duplicate data points are removed.

Classifier	With Duplicates	Without Duplicates
	train set: 2522, testset: 1269	train set: 1772, testset: 990
MultilayerPerceptron	88.42%	91.41%
Logistic	85.03%	89.29%
SMO	84.87%	89.90%
KStar	80.61%	82.83%
J48	80.54%	75.45%
RandomForest	80.46%	79.49%
NaiveBayes	78.33%	82.12%
IBk	75.10%	79.49%

Dataset Exeter Landmarks (see Section 3.5.2 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/exeter/landmark).

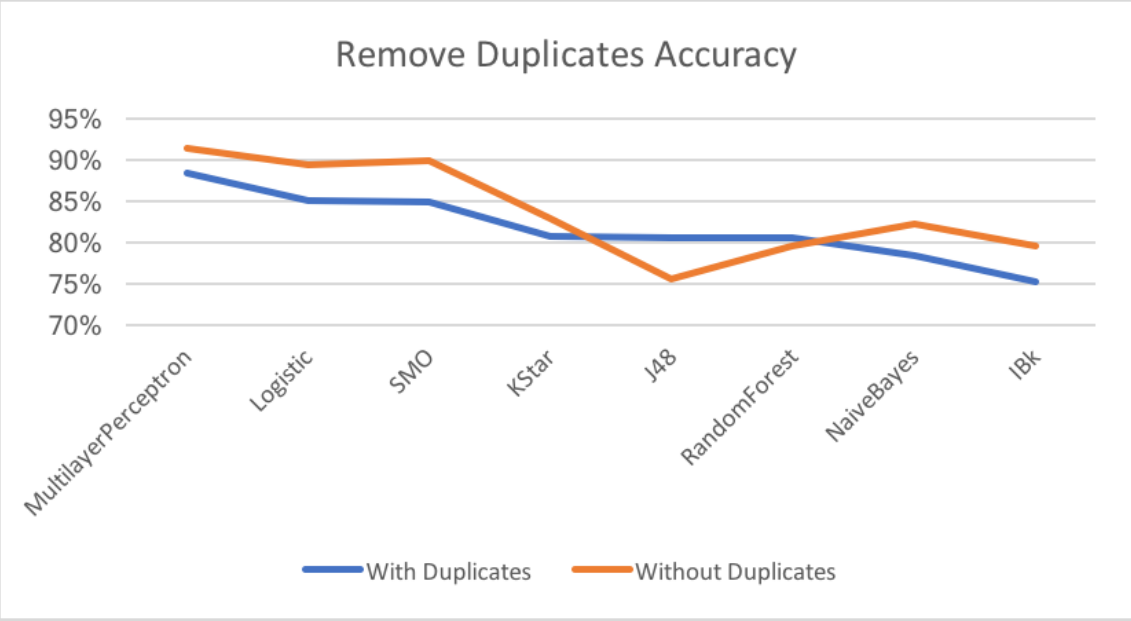

FIGURE 4.1: Landmark prediction accuracy with and without duplicate data points. Dataset Exeter Landmarks (see Section 3.5.2 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/exeter/landmark)

As we can clearly see in Table 4.2 and in a visualized way in Figure 4.1, removing duplicate values increases the accuracy for most of the algorithms and especially for

the ones that perform well. Of course, the computation time is decreased because the ML methods have less data points to consider. As a consequence, for all the following experiments duplicate values are removed.

We can see that after removing the duplicates the Multilayer Perceptron reaches a maximum landmark prediction accuracy of 91.41%.

4.3 Attribute Exclusion

In Section 2.5 we discussed which features seem reasonable to include into the dataset and which should be omitted because they do not seem helpful. In this section we test if the chosen features really all contribute to the prediction accuracy. To observe this, we selectively excluded certain features and studied the influences on the prediction accuracy.

This is done in the Attribute Exclusion Test (on `https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/experiments/AttributeExclusionTest.java`).

4.3.1 Only RSS Values

Accuracy

TABLE 4.3: The accuracy with a different number of RSS values.

Classifier	5 RSS	6 RSS	7 RSS	8 RSS	9 RSS	10 RSS
NaiveBayes	82.05%	79.49%	82.05%	80.77%	84.62%	79.49%
IBk	76.92%	76.92%	82.05%	79.49%	82.05%	75.64%
KStar	75.64%	75.64%	83.33%	78.21%	79.49%	74.36%
SMO	74.36%	70.51%	82.05%	79.49%	78.21%	76.92%
Logistic	73.08%	66.67%	73.08%	66.67%	71.79%	69.23%
RandomForest	71.79%	71.79%	79.49%	75.64%	73.08%	75.64%
J48	64.10%	57.69%	64.10%	64.10%	64.10%	64.10%
MultilayerPerceptron	42.31%	42.31%	43.59%	48.72%	46.15%	44.87%

Dataset Bern Rooms (see Section 3.5.1 and on `https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room`).

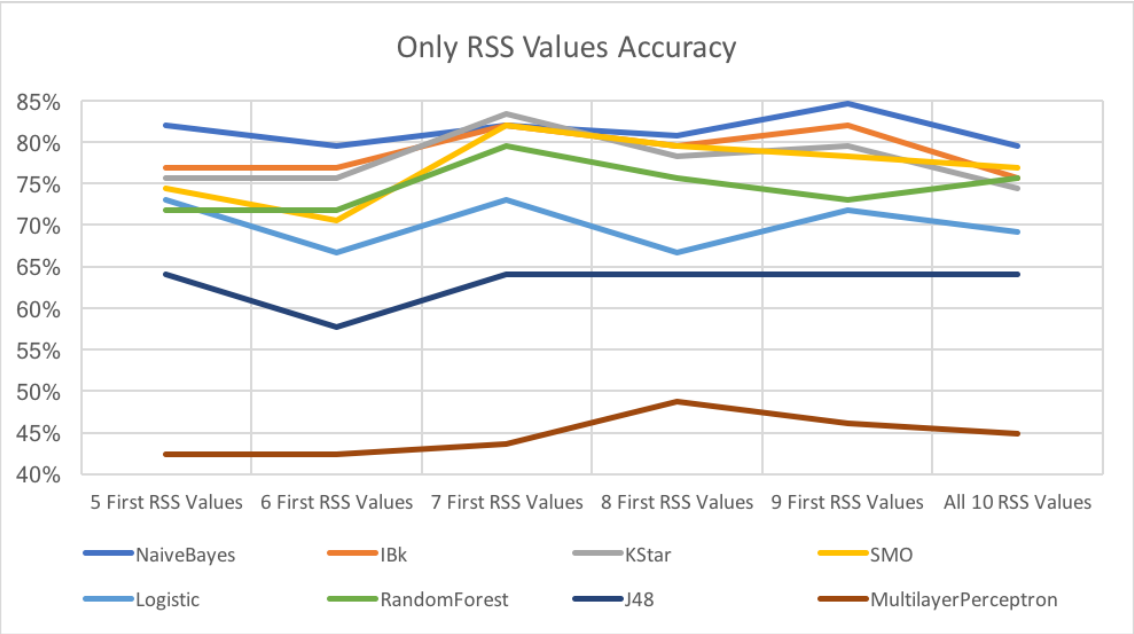

FIGURE 4.2: The accuracy with a different number of RSS values. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room)

Table 4.3 and Figure 4.2 show that there are two peaks, namely when 7 and 9 RSS values are used. The Naive Bayes classifier reaches a maximum prediction accuracy of 84.62% with 9 RSS values. The other methods that perform well (KStar, SMO, Logistic and Randomforest) are considerably better with 7 RSS values. Table 4.4 depicts that there is not a great difference in testing time between 7 and 9 RSS values. Both would be viable options, but because more methods performed well with 7 RSS values and because Naive Bayes was never the best classifier on the final set in our tests, we are working with 7 RSS values in this dataset from now on. We opine that signal interference may be the reason for worse performance when more than 7 RSS values are utilized.

Testing Time

TABLE 4.4: The testing time with a different number of RSS values measured in µs per instance.

Classifier	5 RSS	6 RSS	7 RSS	8 RSS	9 RSS	10 RSS
NaiveBayes	108	201	150	106	83	173
IBk	283	231	262	230	294	297
KStar	863	1747	1095	1139	1285	1633
SMO	58	55	63	45	62	63
Logistic	20	24	21	22	23	28
RandomForest	31	39	46	20	38	34
J48	20	25	59	25	22	23
MultilayerPerceptron	9	10	11	10	7	12

Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

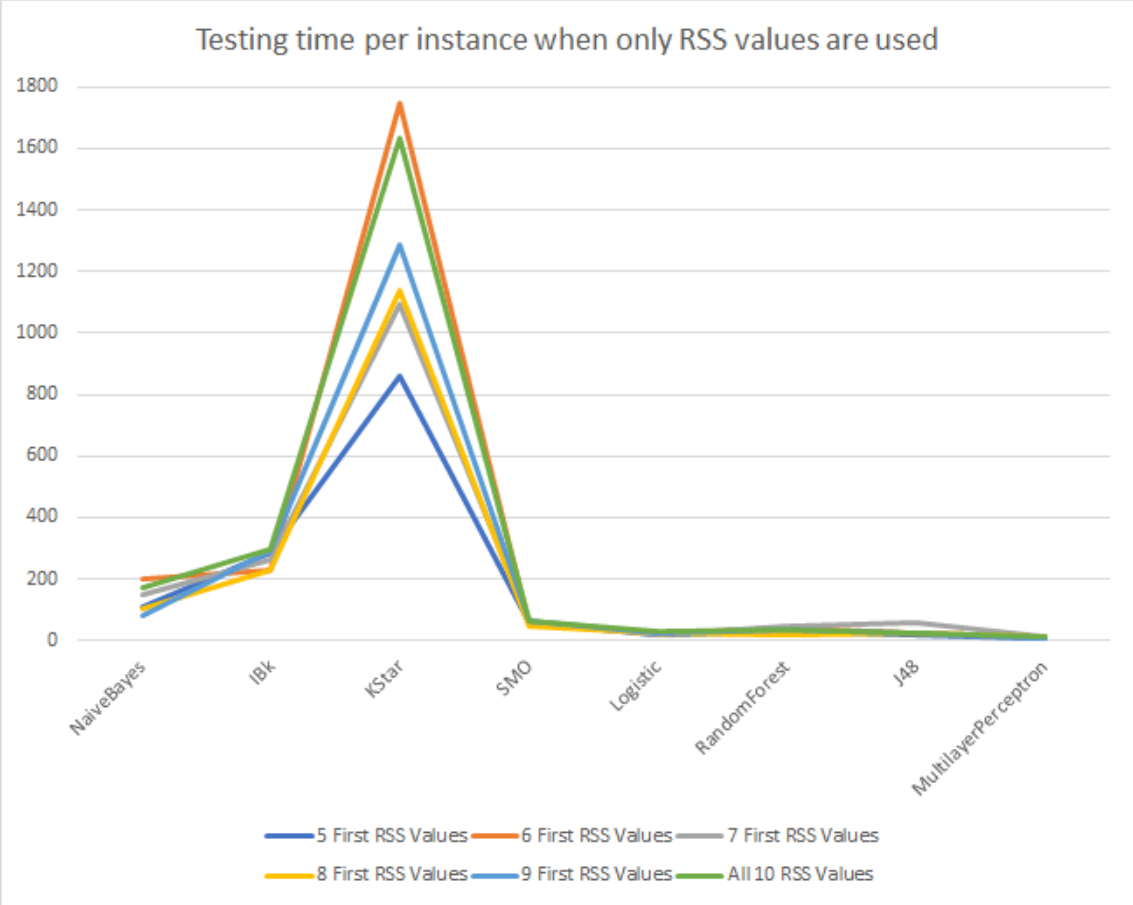

FIGURE 4.3: The testing time with a different number of RSS values measured in µs per instance. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

Testing time is defined as the time needed to classify an instance and is measured in µs per instance. The tests have been conducted on a MacBook Pro, late 2013 edition with a 2.4 GHz Intel Core i5 CPU running macOS Sierra .

Table 4.4 and Figure 4.3 present that instance based methods like IBk (K-Nearest Neighbour) and especially KStar need so much more time than the other methods (IBk between 230 and 297 and KStar between 863 and 1747 microseconds per instance). This is because they have to look at the whole dataset each time an instance is classified, in contrast to other methods that use functions whose parameters are tweaked during the training phase. As a consequence, we excluded KStar from these big datasets (room recognition), because the experiment would take too long, which is not practical in a real scenario. However, for landmark recognition it makes sense to include it, because the datasets are normally much smaller. The best testing time was achieved by the Multilayer Perceptron with predictions made in between 7 and 12 microseconds per instance.

4.3.2 RSS and Magnetic Field

This section describes how the room prediction accuracy changes when we add the magnetic field (see Section 2.5.1), stored in different ways, to the RSS values.

TABLE 4.5: The accuracy with different ways of storing the magnetic field.

Classifier	**1**	**2**	**3**	**4**	**5**
SMO	82.05%	82.05%	87.51%	88.06%	86.98%
IBk	82.05%	82.05%	85.57%	85.21%	83.71%
NaiveBayes	82.05%	82.05%	82.82%	80.94%	79.99%
RandomForest	79.49%	78.21%	78.22%	80.89%	72.25%
Logistic	73.08%	73.08%	78.64%	68.62%	73.21%
J48	64.1%	64.10%	71.27%	71.27%	71.27%
MultilayerPerceptron	43.59%	39.74%	68.30%	68.40%	79.81%

Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).
Legend:
[1] 7 first RSS values
[2] RSS and gravityRaw and magneticRaw
[3] RSS and magneticProcessed
[4] RSS and gravityMagnitude and geomagneticMagnitude
[5] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude

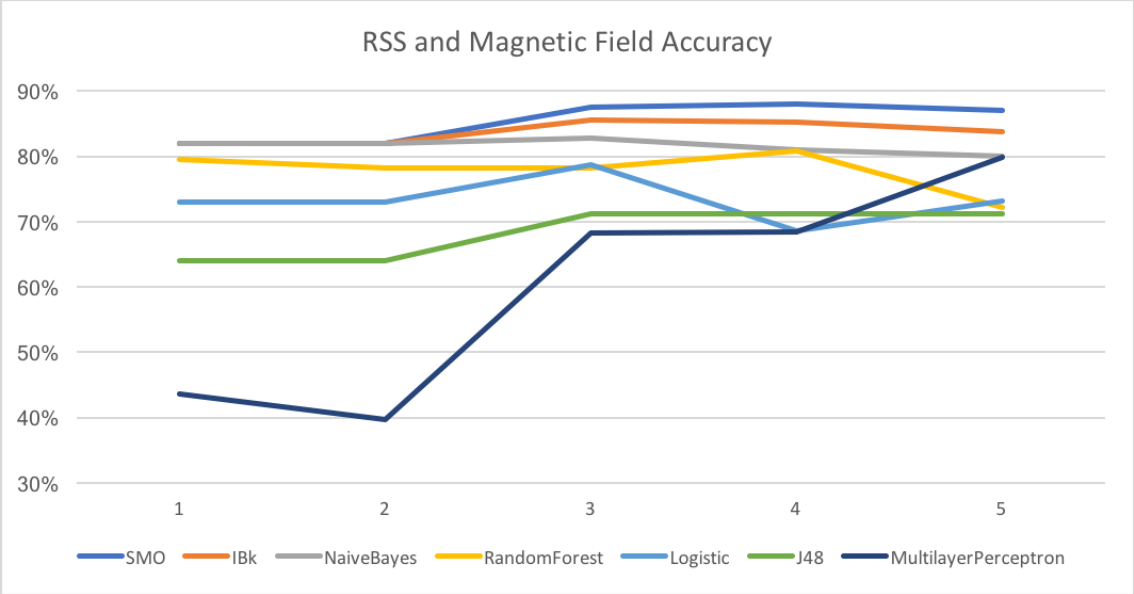

FIGURE 4.4: The accuracy with different ways of storing the magnetic field. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

Legend:
[1] 7 RSS values
[2] RSS and gravityRaw and magneticRaw
[3] RSS and magneticProcessed
[4] RSS and gravityMagnitude and geomagneticMagnitude
[5] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude

Table 4.5 and Figure 4.4 indicate that adding the raw values of the gravity and magnetic field (2) does not improve the accuracy . Adding the magneticProcessed values (3) improves prediction accuracy for most of the algorithms and by 5% for the best performing method SMO (Sequential Minimal Optimization), for example. Adding the gravity magnitude and the geomagnetic magnitude instead of magneticProcessed (4), some methods perform better and some worse. SMO for instance still improves, but Logistic Regression gets worse. Adding both of the last two at the same time, (5) we could expect that the effect is combined. But interestingly, the accuracy of the top performing methods decreases again. On the other hand, the MultilayerPerceptron performs much better. The best room prediction accuracy is reached by the SMO classifier with 88.06% using RSS values, gravityMagnitude and geomagneticMagnitude

4.3.3 Additional Features

This section describes how the room prediction accuracy changes, when we add additional features, namely light and GPS data, to the dataset.

TABLE 4.6: The accuracy with additional features added.

Classifier	1	2	3
SMO	86.98%	88.02%	84.16%
IBk	83.71%	84.74%	81.1%
NaiveBayes	79.99%	90.13%	44.12%
MultilayerPerceptron	79.81%	73.16%	44.93%
Logistic	73.21%	71.81%	55.77%
RandomForest	72.25%	82.33%	41.55%
J48	71.27%	76.57%	35.65%

Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).
Legend:
[1] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude
[2] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude and Light
[3] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude and Latitude and Longitude

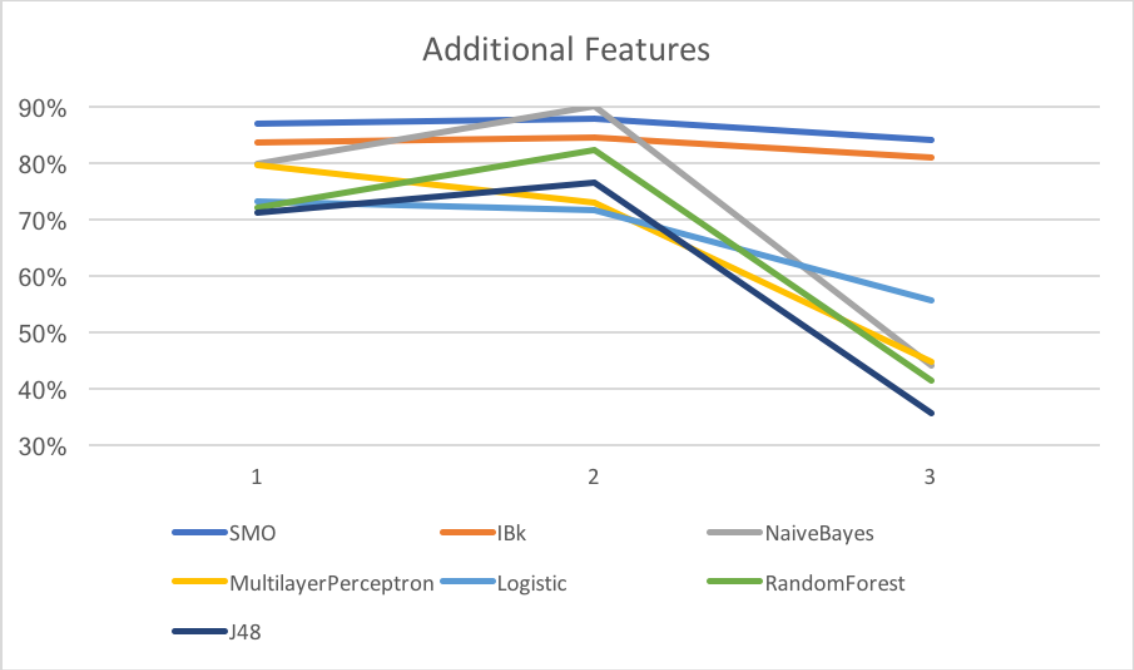

FIGURE 4.5: The accuracy with additional features added. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

Legend:
[1] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude
[2] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude and Light
[3] RSS and magneticProcessed and gravityMagnitude and geomagneticMagnitude and Latitude and Longitude

TABLE 4.7: The confusion matrix for the Naive Bayes classifier using the dataset number 2 (with light). Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

1	2	3	4	5	6	7	8	9	<-- classified as
2145	0	0	0	19	0	0	0	0	1
0	640	0	0	0	0	0	25	0	2
0	104	786	0	0	0	0	0	0	3
0	112	0	137	0	0	0	42	0	4
77	0	45	0	507	0	0	56	0	5
0	0	101	0	0	1217	0	0	0	6
0	73	3	0	0	0	984	0	11	7
0	3	0	116	0	0	0	528	0	8
0	0	0	0	0	0	0	64	829	9

Table 4.6 and Figure 4.5 depict that adding Light comes with an improvement for

most tested methods. The Naive Bayes classifier even reaches an accuracy of 90.13% using RSS values, magneticProcessed, gravitiyMagnitude, geomagneticMagnitude and Light (for more information concerning data collection see Section 3.4).

A confusion matrix is a table with the classification of the ML algorithm in the column and the actual class in the row. So if an instance gets classified as class 2 but actually belongs to class 4, the value at the respective coordinates in the table is incremented by one. This is done for all of the instances in the test set. So on the main diagonal from the top left corner to the bottom right corner there are all the correctly classified instances. On all the other places in the table we can see the incorrectly classified instances. The confusion matrix gives very interesting insights into the problems of the classifiers. It tells us which classes the classifier mixes up. We can see how many instances have been assigned a certain class (column head) by the ML method and what their actual class (row head (to the right)) is. An example for a confusion matrix is shown in Table 4.7.

In the confusion matrix, shown in Table 4.7, it is clearly visible that the classifier has problems with class 2 (the corridor), because many values are not situated on the main diagonal but instead in column and row 2. The classifier often confuses class 2 with the classes 3, 4 and 7. These are some of the smaller rooms, as can be seen in Figure 3.5. It also struggles with predicting class 3 and 8 correctly. Room 6, 7 and to some extent 5 and 9 seem to be 'weak' classes as it (almost) never happens that the classifier selects one of these classes, when it actually was another class. The opposite, namely that the classifier selects another class when it actually was one of these classes, happens sometimes. For the rooms 1 and 2 the opposite applies. A data point which is in one of these rooms very seldom gets classified as another class. So these two seem to be 'strong' classes. The classifier seems to like these classes.

4.4 Rounding

Some sensors return values with more than 5 decimal places although they are less accurate than 0.1. Because of this, we considered rounding these values to the respective measuring accuracy of the sensor in order to reduce the size of the dataset.

The Rounding Test (on `https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/experiments/RoundingTest.java`) checks if the accuracy is increased, when some features containing decimal places are rounded. We performed tests on the dataset reduced to 7 RSS values, magneticProcessed, gravityMagnitude and geomagneticMagnitude. We performed tests where we rounded with accuracy integer, 0.2 and 0.1. However, prediction accuracy did not increase for any of the tests.

4.5 Hyper-Parameter Search

The Hyperparameter Search Test (on `https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/experiments/HyperParameterSearchTest.java` checks if the accuracy is increased when certain parameters from the ML methods are changed. Hyper-Parameter search can be done with algorithms like grid search, multisearch and auto-weka for instance. It can also be done by trial and error. Using these methods we evaluated several classifiers on the train_optimal and test_optimal datasets (on `https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/`

room/). Here we used the first 9 RSS values, the magneticProcessed, geomagneticMagnitude, gravityMagnitude and light features. The three best performing algorithms are the Multilayer Perceptron with an accuracy of 92.08 %, shown in Table 4.8, the Sequential Minimal Optimization with an accuracy of 89.24 %, presented in Table 4.9 and the Naive Bayes classifier with an accuracy of 89.61 %, depicted in Table 4.10.

TABLE 4.8: The confusion matrix for the MultilayerPerceptron classifier. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

a	b	c	d	e	f	g	h	i	<-- classified as
2164	0	0	0	0	0	0	0	0	a = 1
0	527	0	0	0	0	0	0	138	b = 2
0	25	865	0	0	0	0	0	0	c = 3
0	9	0	154	0	0	0	128	0	d = 4
0	43	0	0	642	0	0	0	0	e = 5
0	25	0	0	44	1249	0	0	0	f = 6
0	6	0	0	0	0	1064	0	1	g = 7
0	0	0	149	0	0	0	498	0	h = 8
0	0	0	51	0	0	0	64	778	i = 9

Accuracy: 92.0802 %
Parameters:
weka.classifiers.functions.MultilayerPerceptron -L 0.2 -M 0.1 -N 40 -V 0 -S 0 -E 20 -H a -batch-size 100

TABLE 4.9: The confusion matrix for the SMO classifier. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

a	b	c	d	e	f	g	h	i	<-- classified as
2164	0	0	0	0	0	0	0	0	a = 1
10	632	0	0	0	0	0	0	23	b = 2
0	55	799	0	0	36	0	0	0	c = 3
0	147	0	144	0	0	0	0	0	d = 4
0	43	0	0	642	0	0	0	0	e = 5
204	69	32	0	0	1013	0	0	0	f = 6
0	126	0	0	0	0	945	0	0	g = 7
0	0	0	119	0	0	0	528	0	h = 8
0	0	0	57	0	0	0	7	829	i = 9

Accuracy: 89.2393 %
Parameters:
weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K
"weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator
"weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4"

TABLE 4.10: The confusion matrix for the Naive Bayes classifier. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

a	b	c	d	e	f	g	h	i	<-- classified as
2149	0	0	0	15	0	0	0	0	a = 1
0	393	9	153	0	0	0	110	0	b = 2
0	0	876	0	0	14	0	0	0	c = 3
0	130	0	135	0	0	0	26	0	d = 4
1	0	44	0	628	0	0	12	0	e = 5
0	0	101	0	0	1217	0	0	0	f = 6
0	59	12	0	0	0	973	0	27	g = 7
0	0	0	119	0	0	0	528	0	h = 8
0	0	0	0	0	0	0	64	829	i = 9

Accuracy: 89.6104 %
Parameters:
weka.classifiers.bayes.NaiveBayes

When we analyze the confusion matrices of these three classifers we derive the following observations: the Multilayer Perceptron is very good in column b as compared to the other two; SMO is very good in the triangle above the diagonal; and Naive Bayes has its own problems specially distributed but performs well at other places where the other two are bad (i2 or b5 for instance). This diversity can be used by meta classifiers (ensemble learning methods) which is described in Section 4.6.

4.6 Optimal Prediction With Ensemble Methods

Ensemble methods combine several base classifiers into one in order to improve the prediction accuracy.

The Optimal Prediction Test (on https://github.com/JoelNiklaus/IndoLoc/blob/master/app/src/test/java/ch/joelniklaus/indoloc/experiments/OptimalPredictionTets.java) edits the dataset in a certain way and configures the ML algorithm with parameters in such a way that the accuracy is optimized. This is based on the knowledge out of the previous tests or previous experiences.

We tried the following different ensemble methods: Grading, Stacking, Decorate, Boosting, Bagging, Dagging and RandomSubSpace, all using SMO as the base classifier. However, none of the ensemble learning methods significantly improved the prediction accuracy. In addition, by combining several different classifiers using voting we achieved a better prediction accuracy, as shown in Table 4.11. Voting is an ensemble method which evaluates several different base ML algorithms and then usually combines the results using majority vote.

TABLE 4.11: The confusion matrix for the Majority Vote classifier with three MultilayerPerceptrons, a ClasssificationViaRegression, a RandomSubspace, a LogitBoost, a RandomForest, a Logistic Regression, a SMO and a Naive Bayes classifier. Dataset Bern Rooms (see Section 3.5.1 and on https://github.com/JoelNiklaus/IndoLoc/tree/master/app/src/main/assets/thesis/bern/room).

a	b	c	d	e	f	g	h	i	<-- classified as
2164	0	0	0	0	0	0	0	0	a = 1
0	663	0	0	0	0	0	0	2	b = 2
0	25	865	0	0	0	0	0	0	c = 3
0	32	0	143	0	0	0	116	0	d = 4
0	43	0	0	642	0	0	0	0	e = 5
0	69	0	0	0	1249	0	0	0	f = 6
0	43	0	0	0	0	1027	0	1	g = 7
0	0	0	119	0	0	0	528	0	h = 8
0	0	0	0	0	0	0	64	829	i = 9

Accuracy: 94.0399 %
Parameters:

```
weka.classifiers.meta.Vote -S 1 -B
    "weka.classifiers.functions.MultilayerPerceptron -L 0.4 -M 0.3 -N 100 -V 0 -S 0 -E 20 -H a" -B
    "weka.classifiers.functions.MultilayerPerceptron -L 0.3 -M 0.2 -N 100 -V 0 -S 0 -E 20 -H a" -B
    "weka.classifiers.functions.MultilayerPerceptron -L 0.2 -M 0.1 -N 40 -V 0 -S 0 -E 20 -H a" -B
    "weka.classifiers.functions.MultilayerPerceptron -L 0.2 -M 0.1 -N 40 -V 0 -S 0 -E 20 -H a" -B
    "weka.classifiers.meta.RandomSubSpace -P 0.5 -S 1 -num-slots 1 -I 10 -W
        weka.classifiers.trees.REPTree -- -M 2 -V 0.001 -N 3 -S 1 -L -1 -I 0.0" -B
    "weka.classifiers.meta.LogitBoost -P 100 -L -1.8E308 -H 1.0 -Z 3.0 -O 1 -E 1 -S 1 -I 10 -W
        weka.classifiers.trees.DecisionStump" -B
    "weka.classifiers.trees.RandomForest -P 100 -I 100 -num-slots 1 -K 0 -M 1.0 -V 0.001 -S 1" -B
    "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4" -B
    "weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K
    "weka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007" -calibrator
    "weka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4" -B
    "weka.classifiers.bayes.NaiveBayes"
    -R MAJ
```

We can see that the room prediction accuracy could be improved by almost 2% to 94.0399% over the the best base classifier (the Multilayer Perceptron) with 92.0802%. This is a large improvement on this level.

Looking at the confusion matrices, shown in Tables 4.8, 4.9, 4.10 and 4.11 we can also see that the voting classifier, depicted in Table 4.11, adopted some behaviour of some classifiers and other behaviour of others. In general, for instance, it adopted the good behaviour of the Multilayer Perceptron, depicted in Table 4.8, in column b. However, it does not make the mistake in i2 (column i, row 2) anymore but it adopted the good behaviour of the Naive Bayes classifier, shown in Table 4.10. Unfortunately, it still has the problems at h4 probably originating from the Multilayer Perceptron, shown in Table 4.8. If we gave more weight to the SMOs prediction, shown in Table 4.9, for instance (not having this problem at h4) it would probably resolve this issue but the prediction accuracy in column b would deteriorate again. In general, it can be seen that it seems difficult to distinguish the rooms 4 and 8, which are two small rooms next to each other, as depicted in Figure 3.5.

Chapter 5

Conclusions and Future Directions

In this chapter we conclude the main contributions and findings of this thesis and over the entire project and give ideas for future work in this area.

5.1 Conclusion

In this thesis we first developed an Android app, which is able to collect data of the phone's surroundings. This data comprises of the RSS values of the nearby WiFi access points, of information about the earth's magnetic and gravity field and of sensor information, namely pressure, ambient temperature, relative humidity and light. The application is further able to train several ML methods to distinguish between different rooms and landmarks (specified areas within rooms). Second, we collected data in order to analyze and then optimize the performance of the chosen ML algorithms.

For room recognition in the dataset taken in the INF building in Bern (see Section 3.5.1), distinguishing between 9 different rooms we achieved the following results: A Multilayer Perceptron, the best base classifier, reached an accuracy of 92.08% (see evaluation 4.8). Combining several base learners using Majority Vote (namely three MultilayerPerceptrons, a ClasssificationViaRegression, a RandomSubspace, a LogitBoost, a RandomForest, a Logistic Regression, a SMO and a Naive Bayes classifier) we could reach a maximal accuracy of even 94.0399 % (see evaluation 4.11). For landmark recognition in the dataset taken in student accommodation in Exeter (see Section 3.5.2) distinguishing between 5 different landmarks we achieved 91.41% accuracy using a Multilayer Perceptron.

Because of these very high accuracies in both room and landmark recognition we are confident that this approach using machine learning can improve the ITS.

5.2 Future Directions

5.2.1 Device Independence

In the experiments done in the scope of this project, we collected the training and the testing data sets on the same phone. Different hardware measuring the environment differently could make the accuracy deteriorate vastly. This could be tested with various current smart phones.

One solution for this problem could be data normalization. For instance, we would not store the RSS values directly, but compute the difference to a representative starting point and normalize the result to a value between 0 and 1.

5.2.2 Only Predict if Sure

An idea to improve the security of a prediction would be to only forward a prediction from Indoloc to the ITS if the Indoloc system is sure (probability greater than some defined threshold). In this way we could ensure that almost no wrong predictions are made which could confuse the ITS. But of course, it would also come with less predictions over all. And it still has to be tested if false predictions are (almost) only made when the Indoloc system is not sure.

5.2.3 Further Optimization of HyperParameters

As already discussed, it is very difficult to find optimal hyperparameters for the ML methods. Further testing could be done to optimize these.

5.2.4 Longterm Stability

As described in Section 1.2, no test about longterm stability has been done yet. So it has to be tested if a model trained with data collected at point A is still performing well enough a month, a year or even more later.

5.2.5 Light

As described in the results of Section 4.3.3, the light feature improves the accuracy for most algorithms. But in the night, or if it is a cloudy day, or in a different season or under some other different condition, it probably decreases the accuracy.

A possible solution for this would be the following: we only consider the relative light difference between the different rooms or landmarks respectively.
But of course if we detect that the proportionalities of the light strengths in the rooms or landmarks respectively are not similar under some circumstances it does not make sense to include the light feature at all!

5.2.6 Bluetooth

By using specially installed Bluetooth beacons at the borders of the room (predominantly doors) we hope to further improve the accuracy of the ITS because that is where Indoloc has the greatest problems identifying the correct room.

5.2.7 RSS Values

As described in Section 2.5.1 we collected the RSS values of all the nearby access points. A value lower than -80 is very weak and may suggest that the access point is very far away or there are many or big obstacles between the access point and the collecting device. This means that the access point is of little or no use to the system. Therefore, we could remove access points with values smaller than -80 from the list in a future version.

Bibliography

Aha, D. and D. Kibler (1991). "Instance-based learning algorithms". In: *Machine Learning* 6, pp. 37–66.

Atkeson, C., A. Moore, and S. Schaal (1996). "Locally weighted learning". In: *AI Review*.

Breiman, Leo (1996). "Bagging predictors". In: *Machine Learning* 24.2, pp. 123–140.

— (2001). "Random Forests". In: *Machine Learning* 45.1, pp. 5–32.

Carrera, Jose Luis, Zhongliang Zhao, Torsten Braun, and Zan Li (2016). "A Real-time Indoor Tracking System by Fusing Inertial Sensor, Radio Signal and Floor Plan". In: *IEEE*.

Carrera, Jose Luis, Zhongliang Zhao, Torsten Braun, Zan Li, and Augusto Neto (2016). "A Real-time Indoor Tracking System in Smartphones". In: *IEEE*.

Cessie, S. le and J.C. van Houwelingen (1992). "Ridge Estimators in Logistic Regression". In: *Applied Statistics* 41.1, pp. 191–201.

Chang, Chih-Chung and Chih-Jen Lin (2017). *LibSVM*. URL: https://www.csie.ntu.edu.tw/~cjlin/libsvm/ (visited on 11/02/2017).

Cleary, John G. and Leonard E. Trigg (1995). "K*: An Instance-based Learner Using an Entropic Distance Measure". In: *12th International Conference on Machine Learning*, pp. 108–114.

Deng, Zhi-An et al. (2016). "Continuous Indoor Positioning Fusing WiFi, Smartphone Sensors and Landmarks". In: *Sensors*.

Frank, Eibe, Mark Hall, Geoffrey Holmes, et al. (2010). "Weka-A Machine Learning Workbench for Data Mining". In: *Data Mining and Knowledge Discovery Handbook*. Ed. by Oded Maimon and Lior Rokach. Boston, MA: Springer US, pp. 1269–1277. ISBN: 978-0-387-09823-4. DOI: 10.1007/978-0-387-09823-4_66. URL: https://doi.org/10.1007/978-0-387-09823-4_66.

Frank, Eibe, Mark Hall, and Bernhard Pfahringer (2003). "Locally Weighted Naive Bayes". In: *19th Conference in Uncertainty in Artificial Intelligence*. Morgan Kaufmann, pp. 249–256.

Freund, Yoav and Robert E. Schapire (1996). "Experiments with a new boosting algorithm". In: *Thirteenth International Conference on Machine Learning*. San Francisco: Morgan Kaufmann, pp. 148–156.

Friedman, J., T. Hastie, and R. Tibshirani (1998). *Additive Logistic Regression: a Statistical View of Boosting*. Tech. rep. Stanford University.

Google (2017). *ML Workflow*. URL: https://cloud.google.com/ml-engine/docs/ml-solutions-overview (visited on 11/02/2017).

Hastie, Trevor and Robert Tibshirani (1998). "Classification by Pairwise Coupling". In: *Advances in Neural Information Processing Systems*. Ed. by Michael I. Jordan, Michael J. Kearns, and Sara A. Solla. Vol. 10. MIT Press.

Ho, Tin Kam (1998). "The Random Subspace Method for Constructing Decision Forests". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.8, pp. 832–844. ISSN: 0162-8828. URL: http://citeseer.ist.psu.edu/ho98random.html.

John, George H. and Pat Langley (1995). "Estimating Continuous Distributions in Bayesian Classifiers". In: *Eleventh Conference on Uncertainty in Artificial Intelligence*. San Mateo: Morgan Kaufmann, pp. 338–345.

Keerthi, S.S. et al. (2001). "Improvements to Platt's SMO Algorithm for SVM Classifier Design". In: *Neural Computation* 13.3, pp. 637–649.

Kittler, J. et al. (1998). "On combining classifiers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.3, pp. 226–239.

Kuncheva, Ludmila I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. John Wiley and Sons, Inc.

Li, Zan et al. (2016). "Fine-grained indoor tracking by fusing inertial sensor and physical layer information in WLANs". In: *IEEE*.

Mascharka, David and Eric Manley (2015). "Machine Learning for Indoor Localization Using Mobile Phone-Based Sensors". In: *IEEE*.

— (2016). "LIPS: Learning Based Indoor Positioning System Using Mobile Phone-Based Sensors". In: *IEEE*.

Melville, P. and R. J. Mooney (2003). "Constructing Diverse Classifier Ensembles Using Artificial Training Examples". In: *Eighteenth International Joint Conference on Artificial Intelligence*, pp. 505–510.

— (2004). "Creating Diversity in Ensembles Using Artificial Data". In: *Information Fusion: Special Issue on Diversity in Multiclassifier Systems*. submitted.

MiTAC (2017). *GPS Signal*. URL: http://www.mio.com/technology-gps-signal.htm (visited on 06/12/2017).

Platt, J. (1998). "Fast Training of Support Vector Machines using Sequential Minimal Optimization". In: *Advances in Kernel Methods - Support Vector Learning*. Ed. by B. Schoelkopf, C. Burges, and A. Smola. MIT Press. URL: http://research.microsoft.com/%5C~jplatt/smo.html.

Quinlan, Ross (1993). *C4.5: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann Publishers.

Seewald, A.K. and J. Fuernkranz (2001). "An Evaluation of Grading Classifiers". In: *Advances in Intelligent Data Analysis: 4th International Conference*. Ed. by F. Hoffmann et al. Berlin/Heidelberg/New York/Tokyo: Springer, pp. 115–124.

Shala, Ubejd and Angel Rodriguez (2011). "Indoor Positioning using Sensor-fusion in Android Devices". MA thesis. Kristianstad University.

Ting, K. M. and I. H. Witten (1997). "Stacking Bagged and Dagged Models". In: *Fourteenth international Conference on Machine Learning*. Ed. by D. H. Fisher. San Francisco, CA: Morgan Kaufmann Publishers, pp. 367–375.

Wang, Xi et al. (2016). "An Indoor Positioning Method for Smartphones Using Landmarks and PDR". In: *Sensors*.

Wolpert, David H. (1992). "Stacked generalization". In: *Neural Networks* 5, pp. 241–259.

Xiao, Jiang et al. (2016). "A Survey on Wireless Indoor Localization from the Device Perspective". In: *ACM Comput. Surv.* 49.2, 25:1–25:31. ISSN: 0360-0300. DOI: 10.1145/2933232. URL: http://doi.acm.org/10.1145/2933232.