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

August 3, 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, I 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, I 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 Dr. Zhongliang ZHAO and Jose Luis CARRERA for their help and support during my project. Furthermore, I yield Melissa Chang special thanks for proof reading my thesis and helping me greatly with the English language. Also 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 I am giving a short introduction to the topic and the motivation.

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 houses.

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)

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, I presume that the list of all these values combined is unique at every distinct point in the building.

Furthermore, I 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 I expect these changes to be infrequent.

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

1.3 Machine Learning

In this way, I can collect lots of labelled location data of the building. But 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. So I propose using machine learning methods to make sense of this big chunk of data. By training a classifier on the labelled data collected, rules can be extracted. Feeding in the actual live data of a moving user, the trained classifier can then predict the user's location.

Chapter 2

Background Information and Theory

In this chapter I am introducing the reader to important background knowledge about the topic.

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

The ITS (Carrera, Zhao, Braun, Li, and Neto, 2016; Carrera, Zhao, Braun, and Li, 2016; Li et al., 2016) can predict the user's location based on the previous location of the user and the orientation and movement of the device. Basically, it suggests a cloud 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 Indoloc is to improve the accuracy of the ITS. I 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. 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)

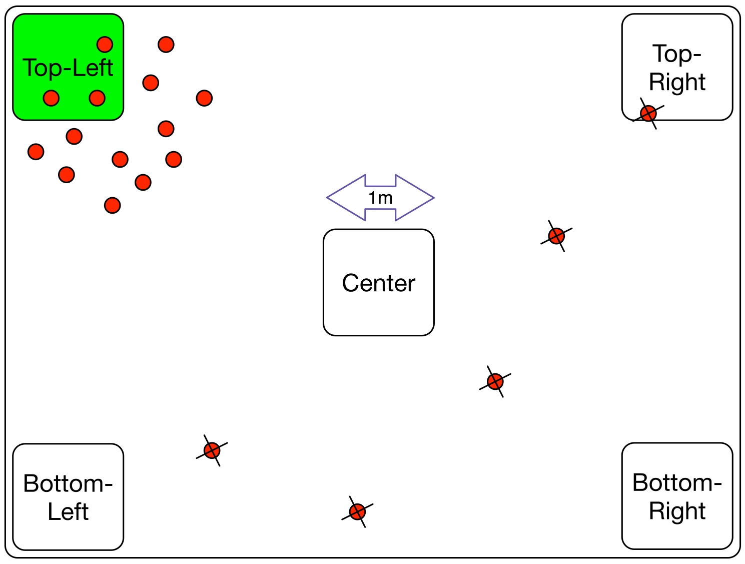

FIGURE 2.1: Five landmarks and the red points predicted by the ITS.

2.2 Reasons for Using Machine Learning

Machine Learning is very suitable for analyzing big amounts of data. The supervised learning methods I am using 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 at the moment and in this way can predict the class (room respectively landmark) of the user.

By using Machine Learning methods I can take advantage of the modern smartphone's ability to collect huge amounts of data about what is happening around them.

2.3 Machine Learning Workflow

In this section I am giving a short introduction about the workflow I used to achieve the best accuracy possible for the tested datasets.

I used both an article about a typical machine learning workflow [1] and the menu of the weka explorer in the graphical graphical user interface of the weka application seen in Figure 2.2 as references for my workflow.

FIGURE 2.2: The menu of the weka explorer in the graphical user interface of the weka application.

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. In the results this is done in section 4.1 and 4.3.

3. **Feature Engineering**
   In this part of the workflow, I 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, I find the best base ML method.

5. **Find Meta Learners**
   In this section, I improve the performance using meta learners. These may combine one or several different base learners (section 4.6).

6. **Hyper-Parameter search**
   Finally, I 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.

[1] Machine Learning Workflow

2.4 Chosen Machine Learning Methods

As Weka provides a very big amount of efficiently implemented machine learning algorithms I could test lots of them with very little effort. An overview can be found on wiki.pentaho.com. The tested ones are listed in the following list. They can also be found in the method `addClassifiers()` in the class AbstractTest.

- Bayes
  - NaiveBayes
- Functions
  - LibSVM
  - Logistic
  - MultilayerPerceptron
  - SMO
- Trees
  - J48
  - RandomForest
- Lazy
  - IBk
  - KStar
  - LWL
- Meta
  - AdaBoostM1
  - Bagging
  - Dagging
  - Decorate
  - Grading
  - LogitBoost
  - RandomSubSpace
  - Stacking
  - Vote

Because an explanation of all the tested algorithms would go beyond the constraints of this thesis I am giving the following recommendations for the interested reader: Good Website to get an overview and Good Article for more details.

2.5 Features

Here both the features used in the system and the ones just taken into consideration are introduced. Each feature corresponds to one column in the dataset and would be referred to as an attribute in Weka.

2.5.1 Used in System

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. They are obtained by scanning the network around the device and then continuously registering the RSS value of each previously specified access point. These values depend on the distance to the access point as well as on the existence obstacles, such as walls or furniture, between the access point and the device.

Magnetic Field

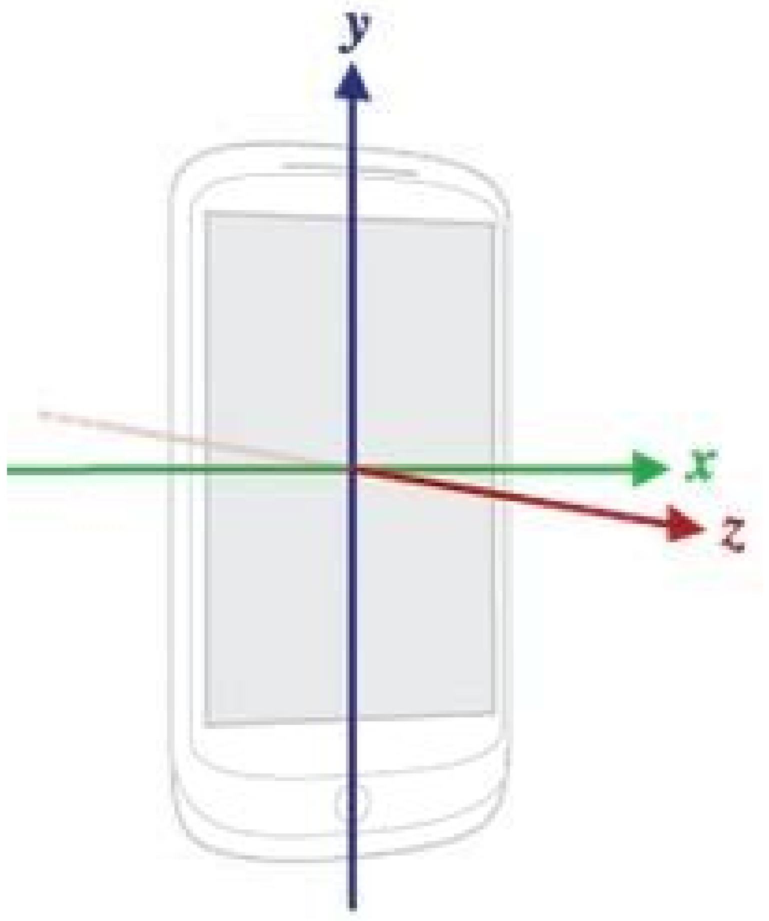

FIGURE 2.3: The values in the device's coordinate system.

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. I 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 I can also derive data from the accelerometer measuring gravity in the device's coordinate system. By using the into

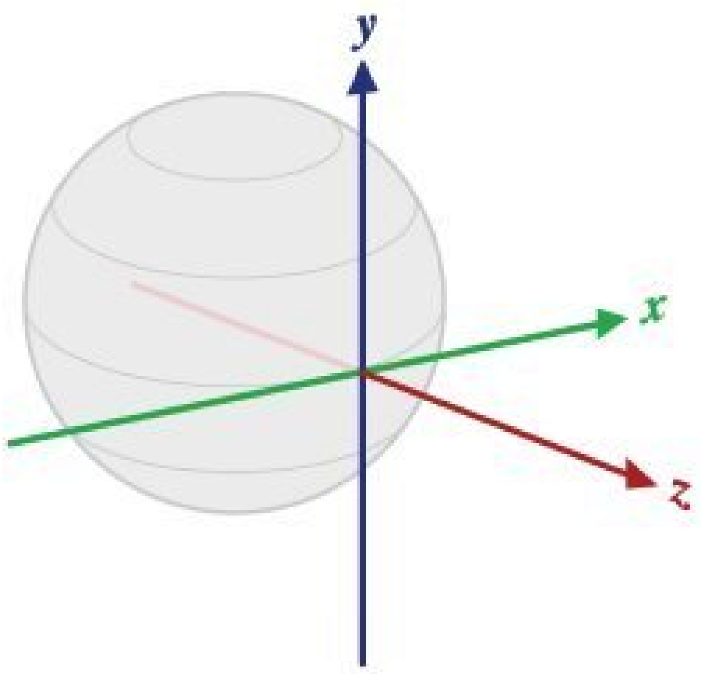

FIGURE 2.4: The values in the earth's coordinate system.

Android built in function `SensorManager.getRotationMatrix()` and providing the magnetic field and gravity values, I can get the rotation matrix `R` and the inclination matrix `I`.

As this is clearly not the way to go, I have two methods of making sense of the raw data (coded in SensorData.java):

MagneticProcessed With the help of `R` I then can do a change of 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 I 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 in North-South-Direction and the z value 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 always is 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) I 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 I 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 methods, I apply a low-pass-filter to the newly collected data. As alpha value I 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 I tested is 0.15 µT. But still the data the sensor outputs contains many more decimal places. In order to exclude the noise, I round the values to 1 µT.

The values obtained by measuring the magnetic field provide an improvement to the overall accuracy.

Light

I also thought about including data collected from the light sensor. Because a room facing a window will clearly be brighter than one surrounded by walls only. And as can be seen in section 4.3.3 this does improve the prediction accuracy.

But 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 for me 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.

I 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 productive environment.

2.5.2 Considerations that have been disregarded

In the following sections the features which have been taken into account but have been judged as not helpful are presented.

Ambient temperature

At first I 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, I 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, I 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 guess that the relative humidity would be higher in the former than the latter. But as with the ambient temperature

I 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 built in a relative humidity sensor.

Pressure

Relative pressure differences are utilised 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 methods 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 here.

Two reduced versions for android [2] [3] have been considered and the one from rjmarsan has been chosen. It uses Weka 3.7.3 and mainly does not include the gui parts of the system which are not used in this application.

2.6.1 Advantages

The main reason why I 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 [4] [5] provides you with some information to get started, the Javadoc [6] is not of much use. It only provides one very small sentence to most of the methods (functions). This sentence mostly does not add

[2]rjmarsan
[3]Shookit
[4]Programmatic Use
[5]Weka in Java Code
[6]Javadoc

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 [7] and a forum on Pentaho [8] where questions concerning Weka can be asked. But unfortunately the active community does not seem to be very large. For my problems at least, I 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 I can recommend the following web resources: Android Developers, Android Studio

2.7.2 Reasons

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 I tested the Android app on the Emulator in Android Studio. This was fine for examining how the user interface behaved but after I added the Wifi, Sensor and GPS collection, I had to test it on real phones. I used the following two Android phones:

Nexus 4 (LG, Android Version 5.0, API, Level 22, Accurate Specifications) and
Moto X Style (Motorola, Android Version 6.0, API Level 24, Accurate Specifications)

[7] Weka Mailing List
[8] Pentaho Forum

Chapter 3

Experimental Setup

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

3.1 General Remarks

The larger the physical gap between rooms respectively landmarks, the larger 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 method to distinguish between the classes.

3.2 Methodology

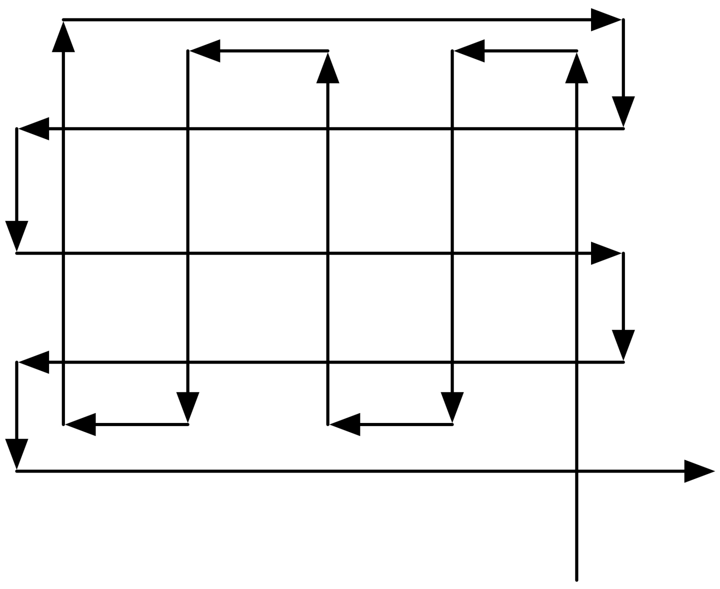

FIGURE 3.1: The grid pattern used to collect the data points.

The data collection for both the training set and the test set have been done in a grid pattern. So I started with the phone in one corner of the room respectively landmark. Then I continuously moved the phone back and forth in parallel vertical lines. Once I covered the entire area in vertical lines I turned the device by 90 degrees and did the same in parallel horizontal lines until I 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.

The training set and the testing set have been collected in two different takes. Because when they were collected in the same take and split up afterwards, Random Forest normally had over 99% testing set accuracy. But in live testing this accuracy was nowhere close (< 70%). The interested reader can see further information in the One Dataset Test in my repository.

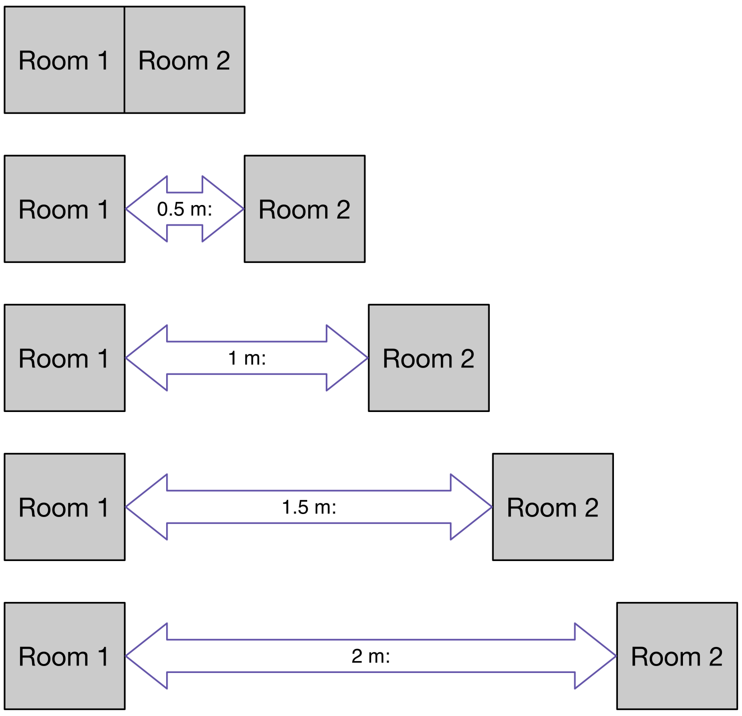

FIGURE 3.2: Distance between rooms.

Generally, the physically closer together the classes (rooms respectively landmarks), the lower the accuracy of the algorithms. This is due to small differences in the measured values.

So on the one hand when the distance between the classes is big, 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.

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. But here I have to be careful, because if the prediction accuracy is not high enough I can assume that a high percentage of the mistakes the prediction makes are close to the border.

Thus, it is difficult to find a good distance between the rooms which gives us a large enough information gain but on which is also reliable enough. Gathering from our experience I need an approximate distance of 1.5 meters between different classes to achieve an accuracy of 85%.

In the method `registerListeners()` in SensorHelper.java I set the collection delay to `SENSOR_DELAY_NORMAL`. Looking at the Android Documentation we read that this delay is set to 200000 microseconds. Therefore the sensor returns at least (!) 5 values per second. So I collected around 5 data points per second walking at a constant rate.

3.3 Room Recognition

In the room recognition phase I 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 in different rooms which are very close to each other. Our best practice was leaving one square metre of space at the doors without any data points collected.

3.4 Landmark Recognition

In the landmark recognition phase I want the system to distinguish several landmarks inside the room. Typically a landmark was of one square metre size and between each of the landmarks I 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) I would have two landmarks, so one at each end. In a normal sized room (around 5x5 metres) I would have four landmarks, one in each corner. In a big room (around 7x7 metres) I would have five landmarks, one in each corner and one in the centre. In our experimental environment there were no bigger rooms available.

3.5 Datasets

3.5.1 Bern

The dataset considered for room recognition has been recorded on the third floor of the CDS (Communication and Distributed Systems) Building of the University of Bern at Neubrueckstrasse 10.

It can be found on my github repository: Room recognition dataset.

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 3.2 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 I 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

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.

It can be found on my github repository: Landmark recognition dataset.

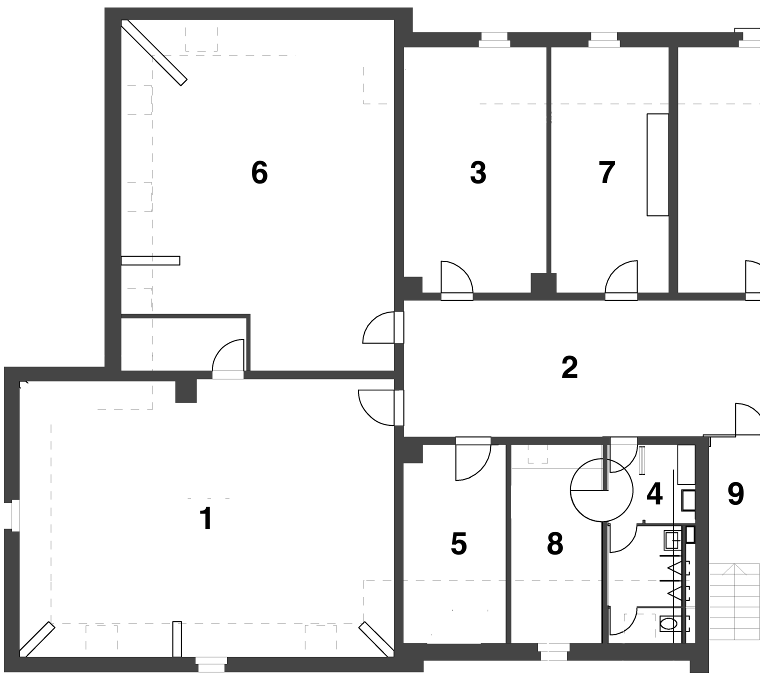

FIGURE 3.3: The testing environment in Bern.

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

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

In this dataset I only collected the magnetic field values in y and z direction and the rss values because at the time of collection I 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 I am confining myself 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.

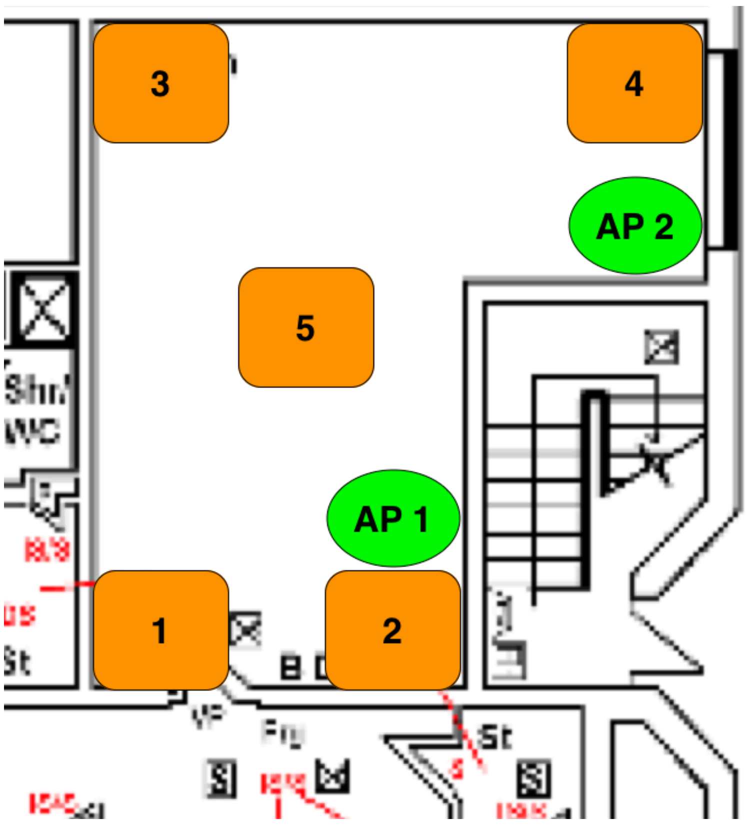

FIGURE 3.4: The testing environment in Exeter

Chapter 4

Discussion of the Experimental Results

In this chapter I am describing the results of the datasets which achieved the best results. I have made the experience that the accuracy reached depends greatly on the environment. On my github repository all the datasets I collected for the experiments are provided for the interested reader.

4.1 Attribute Selection

Running the InfoGainAttributeEval ranking algorithm in weka which evaluates the information gain of each attribute on the Bern Rooms Dataset I get the following result in table 4.1

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 for us. So I 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 Remove Duplicates

The Duplicates Test 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.

As we can clearly see in Table 4.2 and in a visualized way in Chart 4.1, removing duplicate values increases the accuracy for most of the algorithms and especially for the well performing ones. 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 prediction accuracy of 91.41%.

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

Attribute Name	Information Gain	Attribute Index
light	2.354	3
latitude	2.3427	16
longitude	2.138	17
rssValue5	1.949	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

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

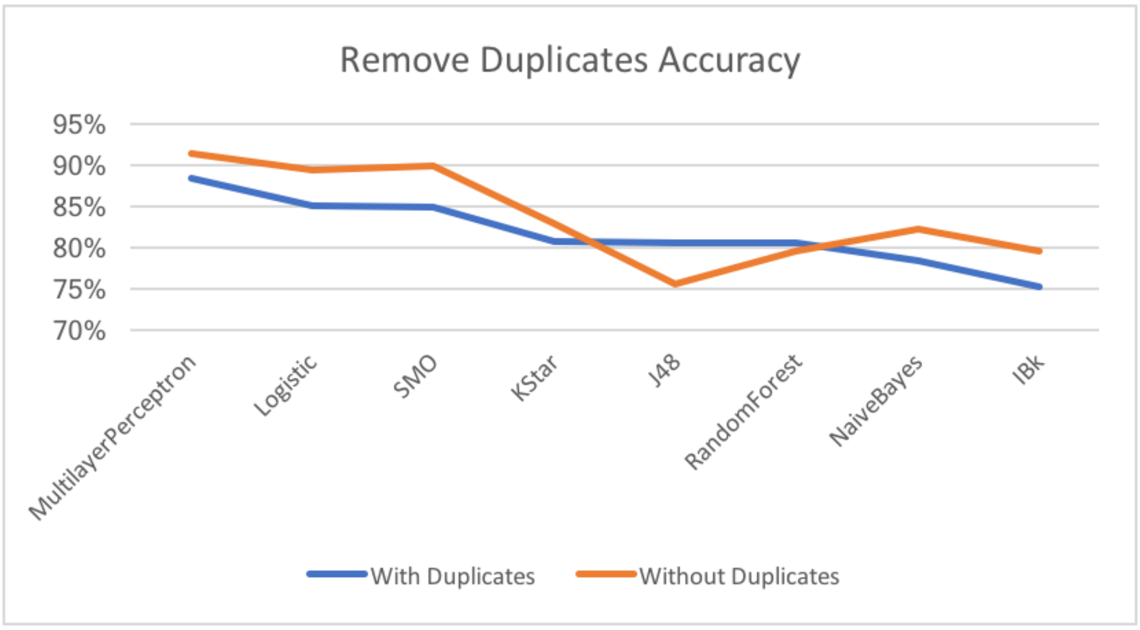

FIGURE 4.1: With and without duplicate data points.

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

4.3 Attribute Exclusion

The Attribute Exclusion Test checks if the accuracy is increased if certain features (attributes) are excluded.

4.3.1 Only RSS values

Accuracy

As we can see in Table 4.3 and Chart 4.2 there are two peaks, namely for 7 and 9 RSS values. 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. 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 my tests I am working with 7 RSS values in this dataset from now on.

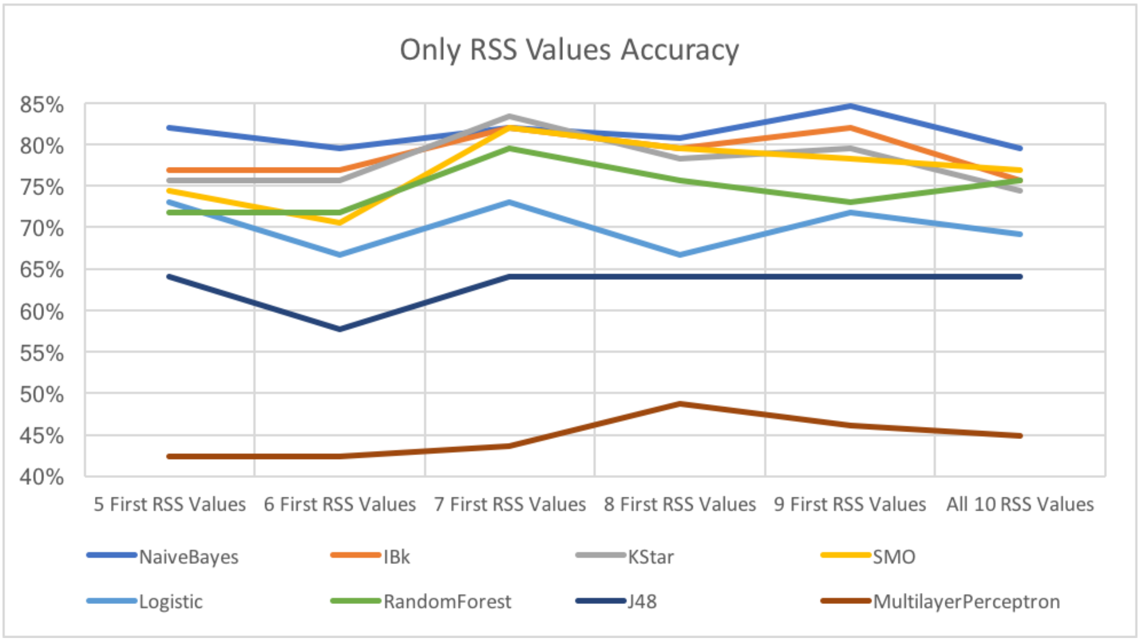

FIGURE 4.2: The accuracy with a different number of RSS values.

Testing Time

As we can see in Table 4.4 and Chart 4.3 instance based methods like IBk (KNN) 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 which use functions whose parameters are tweaked during the training phase. As a consequence, I excluded KStar in these big datasets (room recognition), because the experiment would never end. 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.

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

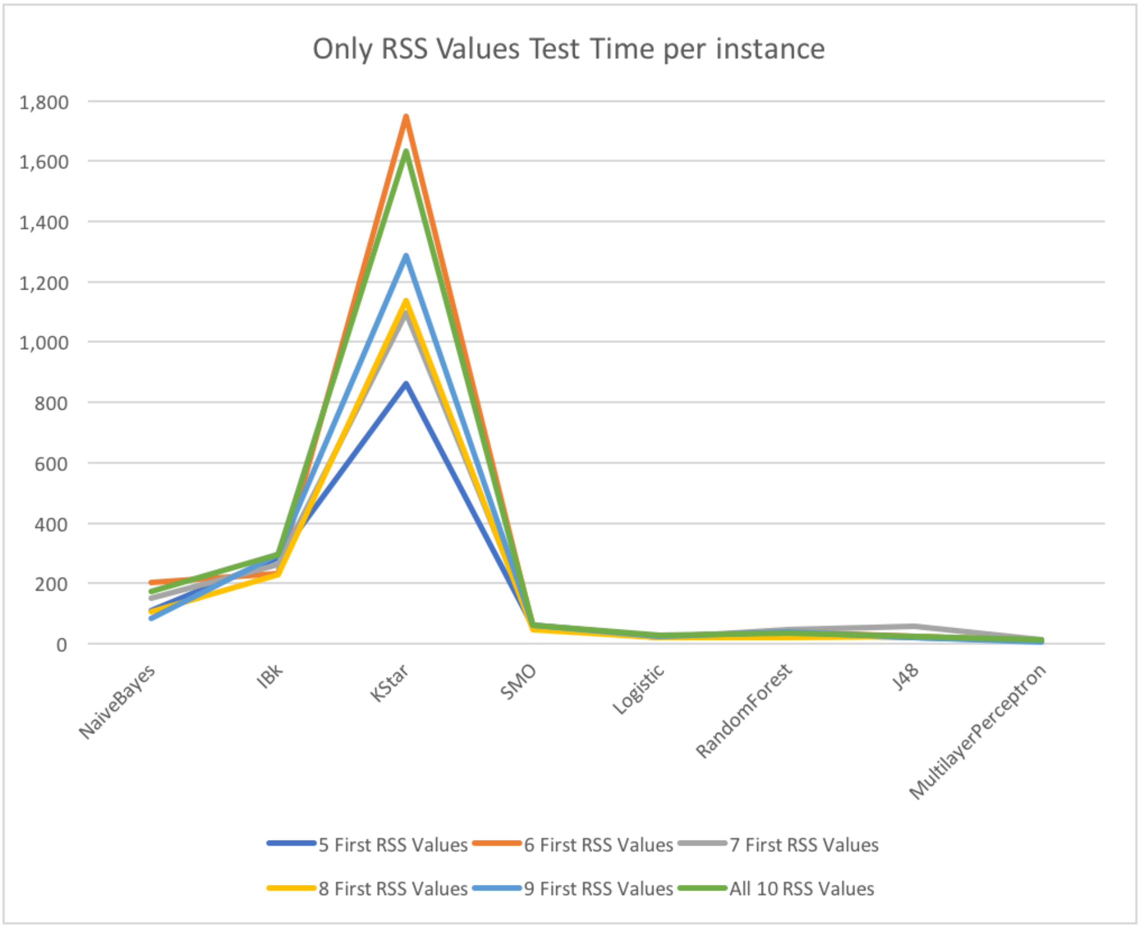

FIGURE 4.3: The testing time with a different number of RSS values measured in µs per instance

4.3.2 RSS And Magnetic Field

As we can see in Table 4.5 and Chart 4.4 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 for instance. 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 prediction accuracy is reached by the SMO classifier with 88.06% using RSS values, gravityMagnitude and geomagneticMagnitude

4.3.3 Additional Features

As we can see in Table 4.6 and Chart 4.5 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. But here I should additionally test if this improvement still holds for a testing set collected in the night for instance (see in conclusion 5.2.5).

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
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

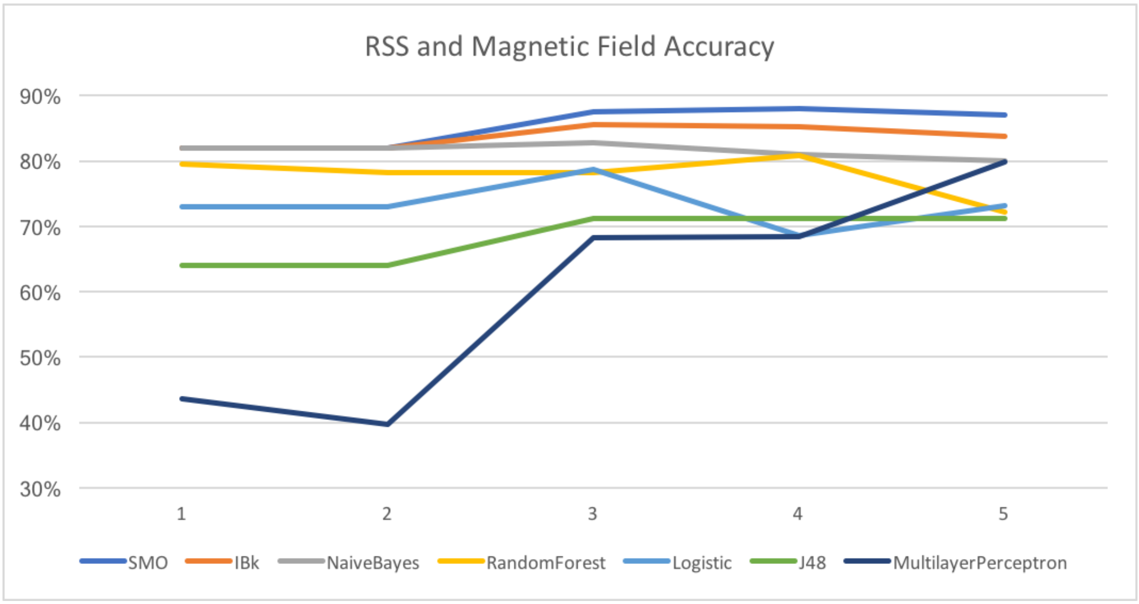

FIGURE 4.4: The accuracy with different ways of storing the magnetic field.

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

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
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)

```
    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
```

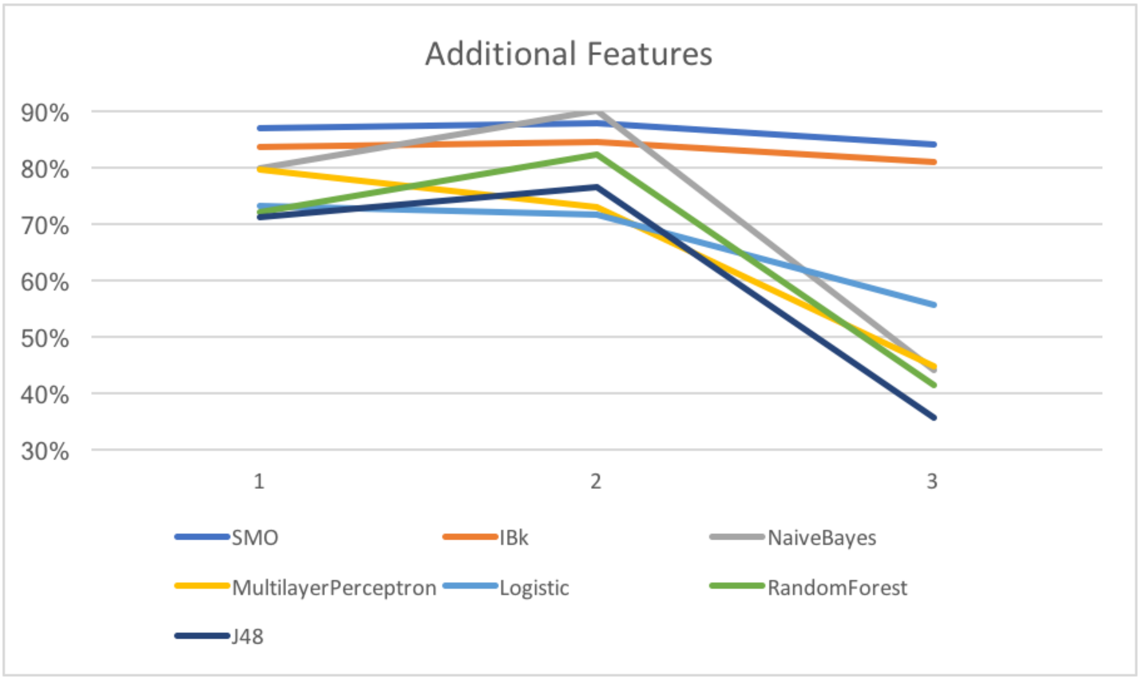

FIGURE 4.5: The accuracy with additional features added.

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

But adding the latitude and longitude values obtained from the GPS seems to only confuse the algorithms apparently. This is probably because the values are just so inaccurate and no more than noise. But the goal of this project is to be independent from GPS indoors.

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. In the confusion matrix 4.7 it is clearly visible that the classifier has problems with class 2 (the corridor). It often confuses it with the classes 3, 4 and 7. These are some of the smaller rooms as can be seen in 3.3. 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 they (almost) never get classified wrongly. But it happens that they are not classified when it would have been this room. For the rooms 1 and 2 the opposite applies. A data point which is in one of these rooms very seldomly gets classified as another class. So these two seem to be ’strong’ classes. The classifier seems to like these classes.

4.4 Rounding

The Rounding Test checks if the accuracy is increased, when some features containing decimal places are rounded. I performed tests on the dataset reduced to the 7

TABLE 4.8: The confusion matrix for the MultilayerPerceptron classifier.

```
   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.

```
   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"

first rss values, magneticProcessed, gravityMagnitude and geomagneticMagnitude. I performed tests where I 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 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, multi-search and auto-weka for instance. It can also be done by trial and error. Using these methods I evaluated several classifiers on the train_optimal and test_optimal datasets. Here I used the first 9 rss values, the magneticProcessed, geomagneticMagnitude, gravityMagnitude and light features. The three best performing methods with its parametrizations were the following:

TABLE 4.10: The confusion matrix for the Naive Bayes classifier.

```
    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: 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. Naive Bayes has its own problems specially distributed but performs well at other places 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

The Optimal Prediction Test edits the dataset in a certain way and configures the ml method with parameters in such a way that the accuracy is maximised. This is based on knowledge out of the previous tests, on experience and on lots of trial and error.

I tried the following different ensemble methods: Grading, Stacking, Decorate, Boosting, Bagging, Dagging and RandomSubSpace all using SMO as the base classifier. I did not succeed in significantly exceeding the accuracy of the base classifier in any of them, but by combining several different classifiers using voting I achieved a better prediction accuracy.

We can see that the 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 we can also see that the voting classifier adopted some behaviour of some classifiers and other behaviour of others. In general for instance, it adopted the good behaviour of the Multilayer Perceptron in column b. However it does not make the mistake in i2 anymore but it adopted the good behaviour of the Naive Bayes classifier. Unfortunately, it still has the problems at h4 probably originating from the MultilayerPerceptron. If I gave more weight to the SMOs prediction 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. This is probably due to their proximity and because they are comparably small rooms.

This is a never ending process though, but given more time it is probably possible to achieve even higher accuracy by tweaking the parameters.

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.

```
   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 -1.7976931348623157E308 -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 ẅeka.classifiers.functions.supportVector.PolyKernel -E 1.0 -C 250007¨-calibrator ẅeka.classifiers.functions.Logistic -R 1.0E-8 -M -1 -num-decimal-places 4" -B "weka.classifiers.bayes.NaiveBayes " -R MAJ

Chapter 5

Conclusion and Future Directions

In this chapter I am drawing a conclusion over the entire project and giving ideas for future work in this area.

5.1 Conclusion

In this project I 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, I 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 CDS building in Bern distinguishing between 9 different rooms I 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) I 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 distinguishing between 5 different landmarks I achieved 91.41% accuracy using a Multilayer Perceptron.

Because of these very high accuracies in both room and landmark recognition I am 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, I 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 smartphones.

One solution for this problem could be data normalization. For instance, I 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 boundary). In this way I 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 said, 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 in 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: I only consider the relative light difference between the different rooms respectively landmarks.
But of course if we detect that the proportionalities of the light strengths in the rooms respectively landmarks 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) I hope to further improve the accuracy of the ITS because that is where Indoloc has the greatest problems identifying the correct room.

Bibliography

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*.

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

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

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

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