Decision tree model for detecting a meeting

Creating a decision tree that calculates the probability of detecting a meeting, with consideration for different properties of the sensor sets

Ernst Fluttert

e.h.fluttert@student.utwente.nl

ABSTRACT
Sensor technology has become an important part of ubiquitous applications. Various sensors can be used to detect human presence and interactions. This paper will present a model to quantify sets of sensors and to present a guideline on different sensor compositions to successfully detect a meeting.

The proposed model takes into account properties such as cost, privacy and accuracy. It also presents a guideline in the form of a decision tree for selecting the most optimal set in terms of the cheapest, the most accurate and the most privacy aware.

Keywords
decision tree, detection, meeting, probability, sensors, sets of sensors, ubiquitous computing

1. INTRODUCTION
There are many interactions between people in an office environment. Various sensors can be used to detect human presence and even human interactions. The diversity of technological solutions makes it interesting to identify a number of sensors and their types, which are required to successfully detect an interaction. Let us consider a meeting as an interaction that has to be detected. We define a meeting as an interaction gathering of at least three people in a room where the door is closed and a presentation or discussion is going on.

Use of a meeting detection system has several benefits and may enable:
- The prevention of interruptions by automatically lighting up a “do not disturb”-sign outside the room;
- A better planning for the meeting room;
- An optimal climate control.

Simply speaking, a ubiquitous system works in the following way: collecting data through embedded sensors and providing assistance in a non-intrusive way, trying to improve the situation [1]. When a system can detect a meeting, it opens the possibility for detecting other kinds of interactions using the same system.

In this paper the focus lies on constructing a method for combining different sets of sensors with the intention to construct different optimal sets in terms of cost, privacy and accuracy. A model will be presented that is capable of calculating the probability for detecting a meeting and deciding on the type of sensors to be used to meet a certain objective.

This paper has been organized as follows: First it will take a closer look at the model and how the probabilities are calculated. To be as flexible as possible, we add parameters to the model that can be modified so the most optimal set will be selected for the inserted parameters. The model can be graphically drawn as a decision tree. Secondly, sensors will be evaluated and will be given values for the parameters that are required for the model. Additional information about each sensor will be presented. Finally, the model is presented as a pseudo code to be used as a base for further research.

1.1 Research questions
In order to create the model there are some key questions that need to be answered. The main question is:

How can one calculate the probability of a sensor used in a combination of sensors, with regard to the different properties of each sensor?

This problem can be diverted into several sub-problems related to:

1. Creating a method of calculating the probability of detecting a meeting for different sets of sensors;
2. Specifying what attributes a sensor should have and assigning sensible values to these attributes;
3. Creating a model that can compare sets of sensors, which includes the use of attributes to compare the results of the model;
4. Visualizing the model in the form of a decision tree.

1.2 Research method
The research will be based on a literature study. This study will be focused on how to create a model and putting this into a decision tree. Another objective will be the mapping of the different attributes of the sensors.

1.3 Literature review
The literature study consists of several parts. The statistical model requires information about decision trees and how these are put into formulas. This kind of literature is mostly focused on very specific (mathematical) problems or has a general view on decision trees. The other broad part of the related work was focused on the sensors, mainly because this study requires information on how the sensors can be used to detect human presence or even human interaction.

2. DECISION TREE
The structure of the decision tree that is used in this paper is resembled best by a binary tree [11]. The method for mapping the multiple sensors in sets into a decision tree is quite simplistic: all nodes correspond to a sensor and the leaf (last node in a branch) holds the information about the set that is represented by a branch. An analogy can be made when mapping a database and its attributes into a decision tree [2]. The decision tree presents an overview on what subsets provide the highest gain in the database.

This paper describes a decision tree where a node represents a sensor; a leaf represents a set of sensors that corresponds with the sensors located in the branch the leaf is in.
There are two branching-factors: the left branch indicates that the sensor is present in the set (is used to detect the meeting); the right branch indicates that the sensor is not used and is excluded.

A node has two properties:

- The probability of detecting a meeting successfully;
- The weight for the combined attributes of the sensor.

The model that will create this kind of decision tree uses heuristic to quantify the probability and the weight of sensor attributes.

To create this model the following steps have been taken:

1. Creating a base model where the goals of the model are defined;
2. Introducing heuristic to quantify sensor attributes and to calculate the probability of the sensor detecting a meeting;
3. Transforming the goals and heuristics into a mathematical model. This mathematical model serves as the base for the decision tree and the pseudo code.

2.1 Base model
When different sensors are used in combination for detecting a meeting, the model should calculate how accurate the composed set is. Besides the accuracy, the model should tell something about other attributes such as cost and privacy. The importance of the attributes should be given by the user.

To achieve this, the model requires:

- A calculation of the probability that a meeting is going on, based on the probability of the sensors used;
- A calculation of the weight of the attributes (cost, privacy and accuracy);
- A calculation of the above for different sets of sensors;
- A display of the optimal set of sensors would be for a given set of sensors and given attribute weights.

2.2 Heuristic
This paper uses heuristic [3] to quantify the accuracy of the sensors and the weighing of the attributes used. The parameters used in the heuristic are attributes, probability and measuring units.

2.2.1 Attributes
There are three attributes for every sensor that have their own weights: cost, privacy and accuracy. This provides a way to compare different aspects of the sensor based on more than probability alone, it is possible that the cost or privacy is more important than probability alone and the optimal set is focused on these attributes. The user should provide these weights; otherwise the weight will be equal for all three attributes. The overview of the attributes of a sensor is shown in Error! Reference source not found.

Table 1: Attributes of the sensors

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cost</td>
<td>How expensive is the sensor? (the cheaper the better)</td>
</tr>
<tr>
<td>Privacy</td>
<td>Is the sensor privacy-aware? (the more privacy-aware the better)</td>
</tr>
<tr>
<td>Accuracy</td>
<td>How accurate is the sensor? (the more accurate the better)</td>
</tr>
</tbody>
</table>

2.2.2 Probability
To calculate the probability of the sensor, 4 properties are considered. These properties must be quantified in order to obtain the probability. The probability factor is defined by three properties that are described in table 2.

The probability of detecting a meeting is strongly dependent on how well the sensor is fit for detecting human presence and interactions. Another factor is the dependency on advanced algorithms to detect human presence. The higher the dependency, the more processing is required. The last property for probability is the measurement accuracy, which defines how accurately the sensor can detect changes in its environment.

Table 2: Properties for the probability

<table>
<thead>
<tr>
<th>Name</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy in measurement</td>
<td>How well can the sensor detect changes in the environment (the higher the accuracy the better)</td>
</tr>
<tr>
<td>Dependency on software</td>
<td>Does the sensor rely on advanced algorithms to determine if there is a meeting? If yes, this means a lot of post-processing is needed (the lower the dependency the better)</td>
</tr>
<tr>
<td>Human detection</td>
<td>How well the output of the sensor can represent the presence of a human (the higher the better)</td>
</tr>
</tbody>
</table>

2.2.3 Measuring units
In order to give a value to all attributes and properties, the model uses two scales.

The first scale will be used to calculate the weight of the attributes of a sensor. These weights can be negative because an attribute can have a negative impact on the result. For example: when choosing a set mainly based on cost (the cheapest set), an expensive sensor should have a negative impact on a set.

The second scale will be for calculating the probability. This scale will go from zero to one in accordance with the range for chances.

Error! Reference source not found., presents an overview of the different scales. To be clear: scale 1 will be used for the attributes of the sensors as described in Table 1; scale 2 will be used for the properties of the probability as described in Error! Reference source not found..

Table 3: Scales for measuring

<table>
<thead>
<tr>
<th>Scale</th>
<th>Rating</th>
<th>Very poor</th>
<th>Poor</th>
<th>Average</th>
<th>Good</th>
<th>Excellent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scale 1</td>
<td>-1,0</td>
<td>-0,5</td>
<td>0,0</td>
<td>0,5</td>
<td>1,0</td>
<td></td>
</tr>
<tr>
<td>Scale 2</td>
<td>0,0</td>
<td>0,25</td>
<td>0,5</td>
<td>0,75</td>
<td>1,0</td>
<td></td>
</tr>
</tbody>
</table>
2.3 Mathematical model

Now that the heuristics are explained and the attributes and properties can be quantified, there is the mathematical model to formalize the two heuristic functions.

First the variables will be defined and then the formulas that the model will use.

2.3.1 Variables

In this section all variables that are used in the model are presented. For each sensor there is a brief explanation and the range of the variable will be specified.

\( S_y \) = Sensor \( y \) with \( y \in \mathbb{N} \)

\( SA_y \) = Sensor \( y \) is present for meeting detection, \( SA \in \{0;1\} \)

\( A_x \) = Attribute \( x \) from sensor \( y \), with \( x \in \{1; 2; 3\} \), and \( A_x \in \{\text{Cost, Privacy, Accuracy}\} \)

\( SC(A_x) \) = Scale of attribute \( x \) of the sensor \( y \), with \( SC \in \{-1, -0.5, 0, 0.5, 1\} \)

\( WA(A_y) \) = Weight of attribute \( x \) of sensor \( y \), with \( WA \in \mathbb{N} \)

\( PA_x \) = Property of probability for sensor \( y \), with \( x \in \{1; 2; 3\} \) and \( PA_x \in \{\text{measurement; dependency; human detection}\} \)

\( SCP(PA_x) \) = Scale of property \( PA \) of sensor \( y \) with \( SCP \in \{0, 0.25, 0.5, 0.75, 1\} \)

\( WPA(PA_x) \) = Weight of the property \( PA_x \), with \( WPA \in \mathbb{N} \)

Note that all variables should be set before using the formulas, which are presented in the next section. This includes weights of the attributes, properties and availability of the sensors. In the pseudo code a matrix will be used where all the required variables have values.

2.3.2 Formulas

Let us start with the formula for calculating the weight of one sensor in the decision tree. After that, the weight of all sensors that are in the branch shall be put in a formula. The weights are based on the attributes of a sensor as described in Table 1.

Weight of sensor \( S = \sum_{y=1}^{3} (SA_y \ast (SC(A_y) \ast WA(A_y))) \)

Weight of branch = \( \sum_{y=1}^{n} \sum_{x=1}^{3} (SA_y \ast (SC(A_y) \ast WA(A_x))) \)

The weight of a sensor is calculated by the summation of all attributes multiplied by the weight of this attribute. The weight of the branch is nothing more than adding up all the weights of the sensors that are in the branch. The weight of all the sensors is used to determine the most optimal set, based on the given attributes.

The other heuristic is used to calculate the probability of successfully detecting a meeting, for a sensor. This probability model is built up from the properties of the probability for a sensor, as described in Table 2.

Probability for sensor \( S = \text{Prob}(S) = \frac{\sum_{x=1}^{3} (SA_y \ast (SCP(PA_x) \ast WPA(PA_x)))}{\sum_{x=1}^{3} (WPA(PA_x))} \)

Probability for a branch = \( 1 - ((1 - \text{Prob}(S_1)) \ast (1 - \text{Prob}(S_2)) \ast \ldots \ast (1 - \text{Prob}(S_n))) \)

The probability of sensor \( S \) is calculated by multiplying all properties with their weights. The result has to be between zero and one because it represents a chance. To achieve this it will be divided by the weight that has been given to the properties of the probability.

Finally the probability of a branch can be calculated by taking all sensors that are in the branch and combining their probabilities \([4]\). The probability of the branch is used to determine the most optimal set, based on probability.

2.4 Output of the model

The model will output two numbers for every set of sensors: probability of detecting the meeting and the weight of the attributes. An optimal set has a high detection probability and a high weight.

An example decision tree based on the proposed model is presented in appendix A, which used three sensors.

3. SENSORS

In this chapter, 7 different sensors are discussed which are often used in a ubiquitous office environment. These sensors are: temperature, humidity, pressure, light, motion, camera, and mobile devices detector.

To use the model, we need to give values to the attributes and properties of probability for every sensor-type. For most of these sensors, information on cost and accuracy can be collected from vendor internet sites such as [http://www.conrad.nl](http://www.conrad.nl) (visited 10 June 2008). When there is no sufficient data about attributes or property of probability for a sensor, a value estimation can be used.

3.1 Temperature sensor

Temperature sensors are widely used for climate control [5] and are most likely to be present in an office environment. Although no research has been done on detecting human presence based solely on the use of a temperature sensor, it is a sensor that can contribute in the process of detecting human presence in combination with other sensor-types [6].

The cost to acquire this sensor is very low, and there are no privacy issues because it cannot detect human presence alone.

The average sensor can measure differences up to 0.1 degrees Celsius, which is good enough if it is for measuring ambient temperatures.

An overview for all variables can be found in Table 4.
3.2 Humidity sensor
Humidity sensors are often combined with temperature sensors, for climate control purposes. Using a well placed humidity sensor, it is possible to detect human presence [6]. This technique requires some calculation to extract this information and has a relatively high false detection, which may be as high as 25% in some cases. Although it can detect human presence, it cannot be used to detect human interaction.

The cost of this sensor is very low. The accuracy is poor because the sensor alone cannot detect a meeting, but it can detect human presence. Privacy is not really an issue because this sensor cannot reveal any sensitive data.

The changes in humidity can be measured with a deviation of 3% relative humidity, which is good enough to detect human presence. For human detection it is not dependent on complex algorithms but it does require some processing [6]. The human detection is rated poor because it is very dependent on proximity to the subject and often less than one meter is required.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Excellent</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Good</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Very poor</td>
<td>-1</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Good</td>
<td>0,75</td>
</tr>
<tr>
<td>Probability: dependency on software</td>
<td>Average</td>
<td>0,5</td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Very poor</td>
<td>0,01</td>
</tr>
</tbody>
</table>

### Table 4: Values of temperature sensor

Table 6: Values of the pressure sensor

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Good</td>
<td>0,5</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Good</td>
<td>0,5</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Average</td>
<td>0</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Good</td>
<td>0,75</td>
</tr>
<tr>
<td>Probability: dependency on software</td>
<td>Excellent</td>
<td>0,95</td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Good</td>
<td>0,75</td>
</tr>
</tbody>
</table>

3.4 Light sensor
Light sensors determine the brightness in the office environment. When the light is turned on or the beamer is being used, it should give some indication of performed activities. Just like a temperature sensor, there has been no research that is focused on detecting human presence based on light sensors. It is reasonable to assume that light sensors could be useful for detection of a meeting in combination with other sensors. For example, turning off the lights, in combination with other sensors that confirm human presence, may indicate the start of a presentation.

The cost of this type of sensor is very low. Because it cannot detect human presence it is not suitable to detect meetings on its own, therefore privacy is not an issue.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Excellent</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Excellent</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Very poor</td>
<td>-1</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Average</td>
<td>0,5</td>
</tr>
<tr>
<td>Probability: dependency on software</td>
<td>Average</td>
<td>0,5</td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Very poor</td>
<td>0,02</td>
</tr>
</tbody>
</table>

3.5 Motion sensor
We consider a motion sensor to be a Passive Infrared (PIR) sensor, which scans for heat emitted by objects (including humans) [8, 9].

These sensors are more expensive than the previously mentioned sensors. The output of the sensor must be interpreted by a good algorithm, thus there is a software dependency. It can detect humans and even detect interactions [8, 9]. The sensor only registers warmth signatures and is difficult to link to persons, in this case privacy is not a major issue.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Poor</td>
<td>-0,5</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Average</td>
<td>0</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Good</td>
<td>0,5</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Good</td>
<td>0,75</td>
</tr>
<tr>
<td>Probability: dependency on software</td>
<td>Poor</td>
<td>0,25</td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Good</td>
<td>0,75</td>
</tr>
</tbody>
</table>

3.3 Pressure sensor
The pressure sensor is a very simple sensor. The sensor requires setting a threshold, which allows an alarm trigger when a weight is put on the sensor and the threshold is passed. The logical location to place this type of sensor is often the chair, e.g. a car seat [7] or a chair in the meeting room.

The cost of the sensor is low. Privacy is not really an issue because a pressure sensor cannot tell who is sitting on it. Pressure sensors can be used to detect human presence, but cannot detect interactions. It scores average on detecting a meeting because it can tell how many chairs are used inside a room.

The measurement for the sensor is dependent on the threshold that is chosen. The sensor itself does not depend on any software.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Excellent</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Excellent</td>
<td>1</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Very poor</td>
<td>-1</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Good</td>
<td>0,75</td>
</tr>
<tr>
<td>Probability: dependency on software</td>
<td>Average</td>
<td>0,5</td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Very poor</td>
<td>0,01</td>
</tr>
</tbody>
</table>
3.6 Camera
A camera does not need to be really expensive for the purpose of detecting interactions of humans [10], a simple camera with only an input of 640x480 pixels can do the job (this equals a webcam).

The cost of a camera is expensive in comparison with some of the previous sensors. A camera is best suited for detecting interactions such as a meeting [12], but this comes with a price. Not only is the cost high, but the privacy is also a really big issue. It is not hard to identify a person from a picture so this should get extra attention.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Poor</td>
<td>-0.5</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Very poor</td>
<td>-1</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Good</td>
<td>0.5</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Good</td>
<td>0.75</td>
</tr>
<tr>
<td>Probability: dependency on</td>
<td>Poor</td>
<td>0.25</td>
</tr>
<tr>
<td>software</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Good</td>
<td>0.75</td>
</tr>
</tbody>
</table>

3.7 Mobile device detector
Today every people have multiple mobile devices like: GSM, PDA and laptop. The technologies used by these devices include GSM, Bluetooth, Wi-Fi and infrared.

Detecting mobile devices requires adongle for every technology, which makes it a rather expensive solution. These dongles will search for mobile devices in their vicinity. A smooth scan process necessitates that the area is not obstructed by physical barriers such as walls to decrease measurement accuracy. A person could also carry multiple devices.

The privacy aspect will not be a problem because the access point only remembers the ID of the device (MAC address) that is trying to connect. However, an association between ID (MAC address) and the persons will lead to highly privacy sensitive data.

<table>
<thead>
<tr>
<th>Name</th>
<th>Rating</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute: Cost</td>
<td>Very poor</td>
<td>-1</td>
</tr>
<tr>
<td>Attribute: Privacy</td>
<td>Good</td>
<td>0.50</td>
</tr>
<tr>
<td>Attribute: Accuracy</td>
<td>Average</td>
<td>0</td>
</tr>
<tr>
<td>Probability: in measurement</td>
<td>Average</td>
<td>0.5</td>
</tr>
<tr>
<td>Probability: dependency on</td>
<td>Average</td>
<td>0.5</td>
</tr>
<tr>
<td>software</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Probability: human detection</td>
<td>Poor</td>
<td>0.25</td>
</tr>
</tbody>
</table>

4. PSEUDO CODE
The pseudo code in this paper uses the JAVA similar syntax, this is because it is object-oriented and the syntax is very clear when reading the code. Using pseudo code has the advantage of clearing what is expected [11].

The code has been divided in three pieces: the required input; the pseudo code itself and the result of the code. The code can be found in appendix B.

4.1 Required input
To build the decision tree there are three inputs required:

- Values for all sensors;
- Weights for the attributes;
- Minimum probability and minimum weight of the leaf.

The values of the sensors should be presented in a matrix where every row represents a sensor and contains the following items: (The numbers before the items are the place in the row.)

1. Rating for cost
2. Rating for privacy
3. Rating for accuracy
4. Sensor available (zero for no, one for yes)
5. Rating for probability: measurement
6. Rating for probability: dependency of software
7. Weight for probability: measurement
8. Weight for probability: dependency of software
9. Weight for probability: human detection

In this way all variables of the model are covered and everything can be calculated. The last variables (7-9) can be altered but for this model they are assigned a value 1.

The part that has huge impact on the model is the attributes weights. Users can play with different values for these attributes.

The last input is that of the desired minimum probability and minimum weight. The user can set these limitations for the decision tree to make the selection of leaves that have the desired aspects. These are not specified in the model because they have no function in the heuristic. Minimum probability has to be between 0 and 1. The minimum weight has to be a whole number (like -5, 0 and 9).

4.2 The model itself
The model has three classes: DecisionTree, Leaf and Sensor. Leaf and Sensor are helper classes that are controlled by the class DecisionTree. The sensor class defines the sensor and its corresponding weight and probability. The leaf class keeps track of which sensors are included in the branch. It will also contain the probability and weight for this branch.

DecisionTree is the class that builds the tree. It requires the input of the sensor matrix with all the values. When the class is initiated it can calculate as many decision trees as the user wants. The function ‘buildTree’ needs 5 parameters to create a tree: weight of cost; weight of privacy; weight of accuracy; minimum weight of the branch and minimum probability of the branch. This function triggers a series of other functions:

1. initTree : clears all variables
2. fillTree : recursively fills the tree until there are no more sensors to process
3. calculateLeaves : calculate the probabilities and weights for the leaves of the tree
4. filterResult : filters the leaves on limits put up by the user and creates the result matrix

Finally, the result will be returned, which will be discussed in output.
4.3 Output
When the algorithm has finished to process all the leaves and selecting the sets that comply the minimum weight and probability is has to output these sets. The model allows to return more than one leaf (set), as long as they comply to the minimum weight and probability. In that case there are multiple interpretations of the optimal set: highest probability to detect a meeting; highest score on the attributes; the number of sensors involved.

The output will be put in a matrix, this matrix has three rows:

1. Best probability
2. Best weight
3. Most sensors

In the first row the sets are sorted on probability: the highest probability will be the first in this row. In the second row the sets are sorted on weights: the highest weight will be the first in this row. In the third row the sets are sorted on the number of sensors the sets contains: the sets with the most sensor will be first.

If a leaf is in the output matrix then it fulfills the minimum requirements that were put in by the user. All leaves are converted to arrays with the following order:

- Accuracy
- Weight
- Total number of sensors

Example: output [1][0] will output an array for the leaf with the best weight in the form of array(accuracy, weight, number of sensors).

The goal of the model is to provide a way to compare sets of sensors. The aim is that the user can specify what aspects are important: cost; accuracy and privacy. With the presented model the user can play with these aspects.

5. CONCLUSION
The paper presents a way to quantify different aspects of various sensors often used in a ubiquitous office environment. It also provides a method of determining the optimal set of sensors to be used in the office environment to detect a meeting. This is only a model and not a proven method, for this reason pseudo code has been added so this model can be better understood and more easily implemented.

The application for the model could lie in designing new office-buildings where the designer wants to have optimal sets of sensors installed. This certainly could be a tool to help him with that.

6. DISCUSSION
The aspects of the sensors in this paper are purely focused on detecting a meeting. It does not only quantify different types of sensors but also compares them in sets. More research is needed to investigate use of various sensors in combination and this work is a small step.

The interesting part is that although the model is built for a meeting detection example, it can also be used in a more general terms for other scenarios where similar sensors are used.

ACKNOWLEDGEMENTS
I would like to thank Nirvana Meratnia; Leon Chevalking and Herwin Weststrate for their feedback on this paper.

REFERENCES
APPENDIX A: EXAMPLE DECISION TREE

Example tree with three sensors.

```
[[Input matrix]]
0 1 2 3 4 5 6 7 8 9 (position in row)
-0.5 0 0.5 1 0.75 0.25 0.75 0 1 1 (motion sensor = sensor 1)
0.5 0 0.5 0 1 0.75 1 0.75 0 1 1 (pressure sensor = sensor 2)
1 1 -1 1 0.75 0.5 0 1 1 1 1 (temperature sensor = sensor 3)
1 1 0.5 0 0.75 0.75 0.25 0 1 1 1 (humidity sensor, should NOT be included)
```

3 sensors are included in the tree, this will result in \(2^3 = 8\) leaves.

Input from user is:

```
buildTree(2, 1, 4, 0, (0.9))
```

Houristic function output per sensor:

- **Sensor 1**
  - Weight = \(-1 + 0 + 2 = 1\)
  - Probability = \((0.75 + 0.25 + 0.75) / 3 = 0.58\)

- **Sensor 2**
  - Weight = \(1 + 0.5 + 0 = 1.5\)
  - Probability = \((0.75 + 1 + 0.75) / 3 = 0.83\)

- **Sensor 3**
  - Weight = \(2 + 1 + -4 = -1\)
  - Probability = \((0.75 + 0.5 + 0) / 3 = 0.42\)

**Most accurate**
1: Leaf 1 (0.96)
2: Leaf 2 (0.93)
3: Leaf 5 (0.90)
4: Leaf 6 (0.83)
5: Leaf 3 (0.76)
6: Leaf 4 (0.58)
7: Leaf 7 (0.42)
8: Leaf 8 (0.00)

**Best weight**
1: Leaf 2 (2.5)
2: Leaf 1 (1.5)
3: Leaf 6 (1.5)
4: Leaf 4 (1)
5: Leaf 5 (0.5)
6: Leaf 3 (0)
7: Leaf 9 (0)
8: Leaf 7 (-1)

**Best sensor combination (biggest set first)**
for accuracy > 0.9 and weight > 0
1: Leaf 1 (Sensor 1 + Sensor 2 + Sensor 3)
2: Leaf 2 (Sensor 1 + Sensor 2)
3: Leaf 5 (Sensor 2 + Sensor 3)
APPENDIX B: PSEUDO CODE

/* class DecisionTree :: Builds a decision tree with regard to the input variables */
public class DecisionTree{
    matrix<int> input; // the matrix that is given by the user
    matrix<array<Integer>> output; // the matrix for the output
    int weightCost, weightPrivacy, weightAccuracy; // input by user for attributes
    int minWeight; // minimum weight of the leaf (branch of the tree)
    double minProb; // minimum probability of the leaf (branch of the tree)
    int totalWeight // totalweight, used by calculation
    array<Sensor> sensors; // holds all sensor-objects
    array<Leaf> leafs; // holds all leaf-objects

    public function decisionTree(matrix<int> m){ //constructor
        input = m; } //init matrix & end constructor

    /* Function buildTree :: this method is used to calculate a tree with the given parameters */
    public function buildTree(int c, int p, int a, int mw, double mp){ //constructor
        this.weightCost = c; //init variables
        this.weightPrivacy = p; this.weightAccuracy = a; //init variables
        this.minWeight = mw; this.minProb = mp //init variable
        initTree(); //make sure to begin with the clean tree
        fillTree(0); // start filling the tree, upside - down
        calculateLeafs(); // calculate weight and probability of the leafs
        filterResult(); // filter all results and order them likewise
        return output; } // returns the output matrix & end of function

    /* function initTree :: resets all arrays and determines the total weights of attributes*/
    public function initTree(){
        sensors = new array<Sensor>(); leafs = new array<Leaf>(); //init variables
        totalWeight = weightCost + weightPrivacy + weightAccuracy; // end of function

    /* function fillTree :: recursively filling up the tree until the last sensor is processed */
    public function fillTree(int s){ // s is the number of the current sensor
        int oldLeafs = leafs.length; //determine old leafs
        /* create new SensorObject */
        // calculate probability
        double prob = (input[s][sensorAvailable] * (input[s][5] * input[s][8]) + (input[s][6] * input[s][9])) / (input[s][7] + input[s][8] + input[s][9] );
        // calculate weight
        int weight = (weightCost * input[s][cost] ) + ( weightPrivacy * input[s][privacy] ) + ( weightAccuracy * input[s][accuracy] );
        newSensor = new Sensor(s, prob, weight); // create new sensor
        sensors.add(newSensor); //puts the sensor last in the array
        // create new leafs
        for(int i=0; i<oldLeafs; i++) //iterate through old leafs, create 2 new leafs,
            //one with and one without this sensor
            if(s!=0) parentLeafSensors = leafs[i].getSensors();
            else( parentLeafSensors = new array<Integer>; ) //no parentLeaf
            newLeaf1 = new Leaf(parentLeafSensors);
newLeaf2 = new Leaf{parentLeafSensors}.addSensor(sensor);
leafs.add(newLeaf1); leafs.add(newLeaf2);} //end forloop
// delete old leaves
for(int i=0; i<oldLeafs; i++){leafs.remove(i)}
// repeat for all steps
int nextSensor = sensor+1;
if(input[nextSensor] != null){fillTree(nextSensor); } //end function fillTree

/* function calculateLeaf :: Calculates the probability and weight for each leave
(after tree is filled) */
public function calculateLeafs(){
for(int i=0; i<leafs.length; i++){ // go through all leaves
Leaf currentLeaf = leafs[i]; // get the current leaf
int weight =0; // init the weight of the leaf
for(int j=0; j<currentLeaf.sensors.length; j++) { //sums up all weights
weight = weight + (sensors[leafSensor[j]].weight); } // in the leafs
currentLeaf.weight = weight; // set weight
double prob =1; // init probability of the leaf
for(int j=0; j<currentLeaf.sensors.length; j++) { //multiply all negated
prob = prob * (1 - (sensors[leafSensor[j]].prob)); // probabilities
}
}
currentLeaf.prob = prob; // set probability
// end forloop
}
// end function calculateLeaf

/* Filter results and puts them in the output matrix */
public function filterResult(){
array<Leaf> resLeafs = new array<Leaf>; // filtered leafs
for(int i=0; i<leafs.length; i++){ // go through all leaves
if(leafs[i].weight > minWeight && leafs[i].prob > minProb){
array<Integer> resultLeaf = array(leafs[i].prob, leafs[i].weight, leafs[i].getNumberSensors)
resLeafs.add(resultLeaf);}
}
resLeafs.sortOn("prob"); // sorts on the property prob
addToResult(resLeafs, 0);
resLeafs.sortOn("weight"); // sort on the property weight
addToResult(resLeafs, 1);
resLeafs.sortOn("sensors"); // sort on the number of sensors involved
addToResult(resLeafs, 2);
}
// end function filterResult

/* function addToResult :: puts leaves into a matrix sorted in a order */
public function addToResult(array<Leaf> resLeafs, int row){
for(int i=0; i<resLeafs.length; i++){
output[row][i]=resLeafs[i]; // add the same order into the output matrix
}
// end function addToResult
}
// end Class DecisionTree
/* class Sensor :: represents a Sensor within the tree*/
public class Sensor{
    int sensorsnumber;      // the number of the sensor
    double prob;            // the probability (accuracy) of this sensor
    int weight;             // the weight of this sensor

    public Sensor(int sensornumber,double prob, int weight){
        this.sensorsnumber = sensornumber; // init the number of the sensor
        this.prob = prob;                 // init probability
        this.weight = weight;             // init the weight
    }
}

/* class Leaf :: represents a branch or leaf within the tree */
public class Leaf{
    array<Integer> sensors; // array that holds the numbers of the sensors that are in the leaf
    double prob;            // the probability (accuracy) of this leaf
    int weight;             // the weight of this leaf

    public Leaf(array<Integer> sensors){ //constructor of this class
        this.sensors = sensors;          // copy the array into the class
    }

    public addSensor(int sensornumber){ // function that adds a sensornumber to the leaf
        sensors.add(sensornumber);      // the number is put at the end of the array
    }

    public getNumberOfSensors(){ // function that returns the amount of sensors
        return sensors.length;        // that are present in this leaf
    }
}