Government of Russian Federation
FEDERAL STATE AUTONOMOUS EDUCATIONAL
INSTITUTION OF HIGHER PROFESSIONAL EDUCATION
National Research University
Higher School of Economics

FACULTY OF COMPUTER SCIENCE
Bachelor's programme
APPLIED MATHEMATICS AND INFORMATICS
Department of Data Analysis and Artificial Intelligence

COURSE WORK

Data acquisition from mobile sensors

Bachelor's student 143 AMI
Kashkinov Matvey Il'ich

Scientific Supervisor
PhD, Assistant Professor,
Attila Kertesz-Farkas

Moscow 2016

Abstract

Our goal is to use data from Android phone sensors to determine whether the device is located indoors or outdoors. To achieve this we have decided to use existing machine learning methods. In this study we describe the process of gathering data and choosing from a wide range of possible approaches to analyze it to reach the best result.

Contents

1. Introduction

Nowadays smart phones have become an integral part of our life. That means that we carry a number of different sensors in our pockets. If properly gathered and processed, the data from those sensors can be used to solve interesting tasks.

The purpose of this study is to implement an application for an Android device that could be used to predict whether it is located inside or outside of the building.

The structure of this paper is organized as follows. The second section presents a thorough presentation of the application interface with screen shots and elementary manual. In section 3 we provide the description of the process of gathering data. We have considered some popular machine learning methods such as Decision Trees, Logistic regression and Support Vector Machine which are described in detail in section 4 that also contains information about how the data was preprocessed.

2. Application description

2.1. Implementation details

The main aim of this work is to create an Android application that can be used to predict whether the phone is located inside or outside of the building.

The secondary task was to find a method to gather the data from the sensors of the Android device. Both of these functions were implemented in the application.

We used Android Studio 2.1 environment and Lenovo A606 as a testing device.

2.2. User interface

Application interface is presented on the screen shot below. Main area contains current readings from all the sensors:

- Accelerometer

  Shows acceleration on three axes

- Proximity

  Shows 0 if there is an object right in front of the face of the device and 1 otherwise

- Light

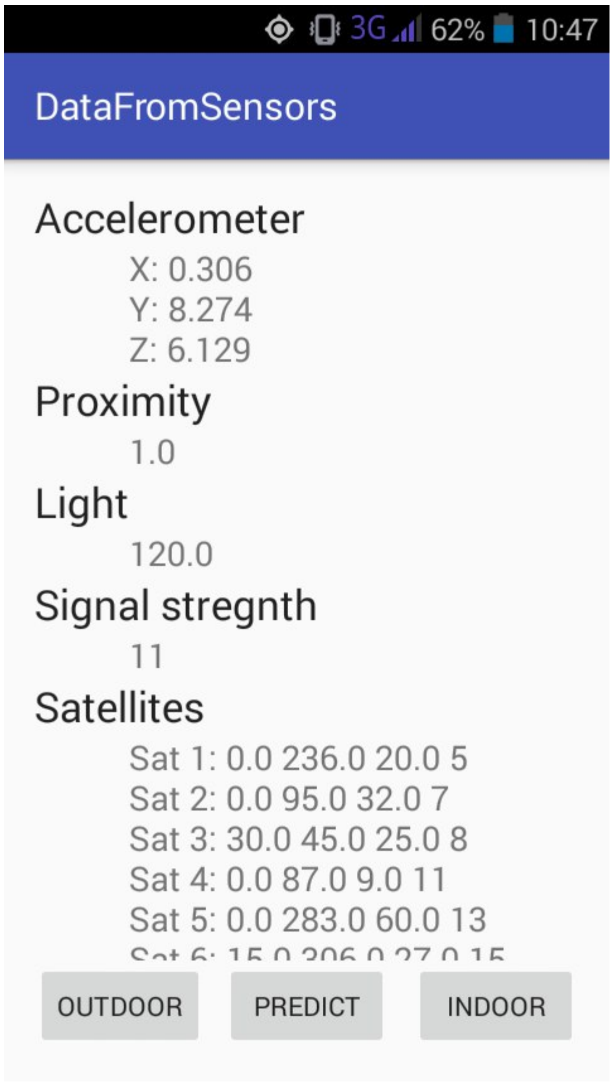
  Shows light strength

- Signal Strength

  Shows mobile networking signal strength

- Satellites

  For each visible GPS satellite shows signal-to-noise ratio, azimuth, elevation and unique pseudo random number



3G 62% 10:47

DataFromSensors

Accelerometer
X: 0.306
Y: 8.274
Z: 6.129

Proximity
1.0

Light
120.0

Signal stregnth
11

Satellites
Sat 1: 0.0 236.0 20.0 5
Sat 2: 0.0 95.0 32.0 7
Sat 3: 30.0 45.0 25.0 8
Sat 4: 0.0 87.0 9.0 11
Sat 5: 0.0 283.0 60.0 13

Figure 1. Interface

In the footer of the application window three button menu is located:

- Predict

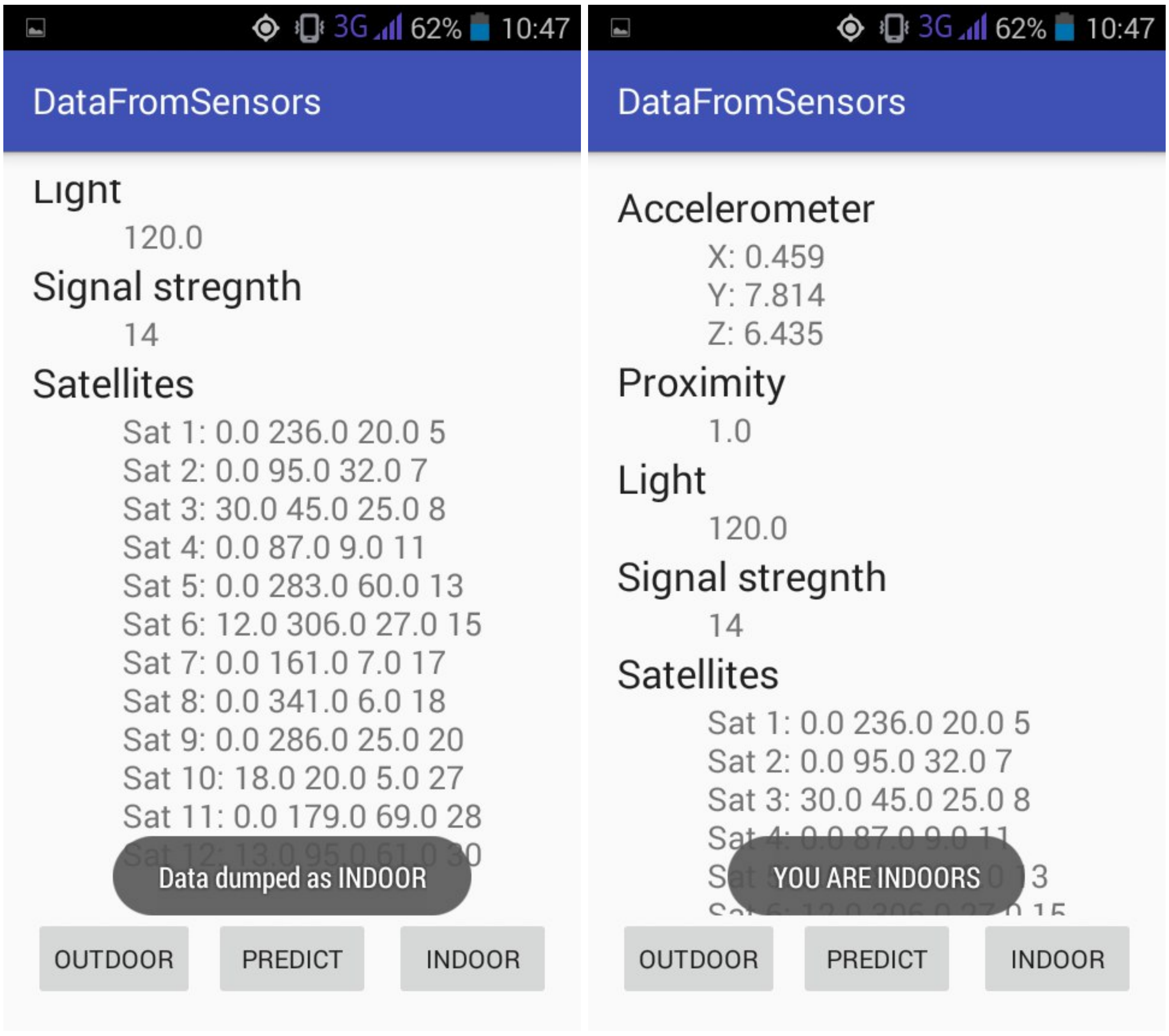
  When clicked shows a toast message with the prediction of the current location

- Outdoor

  When clicked writes all current readings from the sensors with a 0 class label to the CSV file /sdcard/dataset.csv and shows a toast message "Data dumped as OUTDOOR"

- Indoor

  When clicked writes all current readings from the sensors with a 1 class label to the same file and shows a toast message "Data dumped as INDOOR"



Light
120.0
Signal stregnth
14
Satellites
Sat 1: 0.0 236.0 20.0 5
Sat 2: 0.0 95.0 32.0 7
Sat 3: 30.0 45.0 25.0 8
Sat 4: 0.0 87.0 9.0 11
Sat 5: 0.0 283.0 60.0 13
Sat 6: 12.0 306.0 27.0 15
Sat 7: 0.0 161.0 7.0 17
Sat 8: 0.0 341.0 6.0 18
Sat 9: 0.0 286.0 25.0 20
Sat 10: 18.0 20.0 5.0 27
Sat 11: 0.0 179.0 69.0 28
Sat 12: 13.0 95.0 61.0 30
Data dumped as INDOOR

Accelerometer
X: 0.459
Y: 7.814
Z: 6.435
Proximity
1.0
Light
120.0
Signal stregnth
14
Satellites
Sat 1: 0.0 236.0 20.0 5
Sat 2: 0.0 95.0 32.0 7
Sat 3: 30.0 45.0 25.0 8
Sat 4: 0.0 87.0 9.0 11
YOU ARE INDOORS

Figure 2. Interface

3. Data acquisition

To make any predictions we had to collect data at first. We used Lenovo A606. This phone has only three sensors: Accelerometer, Proximity and Light so at first we decided to gather data from them. Obviously, that was not enough and we required some aditional data. The vast majority of android devices support GPS and every phone can measure reception signal strength. So we added Signal Strength and some GPS data such as GPS coordinates and GPS accuracy.

After a number of tests it was clear that GPS takes too much time to make a fix. That is why we had to gather data from satellites directly instead. Android API allows to receive signal-to-noise ratio for each visible satellite which is useful for our task. Another advantage of that decision was that we could measure signal-to-noise ratio even there were not enough visible satelites to make a fix.

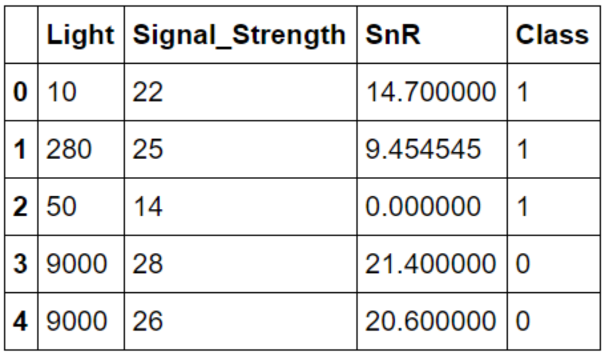
Finally we started collecting data both indoors and outdoors at night and during a day. As described in User Interface section (Section 2.2) collected data was dumped to a CSV file that was then processed on a laptop using python.

4. Data processing

4.1. Preprocessing

We used scikit-learn, numpy, matplotlib and other Python libraries to analyze the data. First we extracted data from the CSV file using Pandas library. Before fitting any of the classifiers data has to be organized into the form of features and classes.

It was obvious that accelerometer and proximity data are useless for our task so we did not take them into account. The next step was to adapt the satellites data. We decided to calculate the average signal-to-noise ratio for each record and use it as a feature. That means that we had three features for each object: Light, Signal strength and Signal-to-noise ratio:



	Light	Signal_Strength	SnR	Class
0	10	22	14.700000	1
1	280	25	9.454545	1
2	50	14	0.000000	1
3	9000	28	21.400000	0
4	9000	26	20.600000	0

Figure 3. Example of the preprocessed data

When the data was ready for fitting the classifier we had to choose which classifier to use. If our data is linearly separable it is preferable to use a linear model for classification that calculates the output score as

$$y = f(\vec{w}, \vec{x}) = f\left(\sum_i w_i x_i\right)$$

where $\vec{x}$ is a vector of features and $\vec{w}$ is a vector of weights.

To check if we can linearly separate our data we can build a scatter plot where axes are features and dots are objects described by their features. As long as we have only three features, it was easy to visualize a 3D scatter plot (Figure 4). Blue dots represent outdoor class whereas red ones stay for indoor class.

It proved our assumption that we can make a hyperplane that would more or less divide our objects into two classes. However this plot revealed another problem. It can be noticed that both red and blue dots are nearly uniformly distributed along Signal Strength axis.

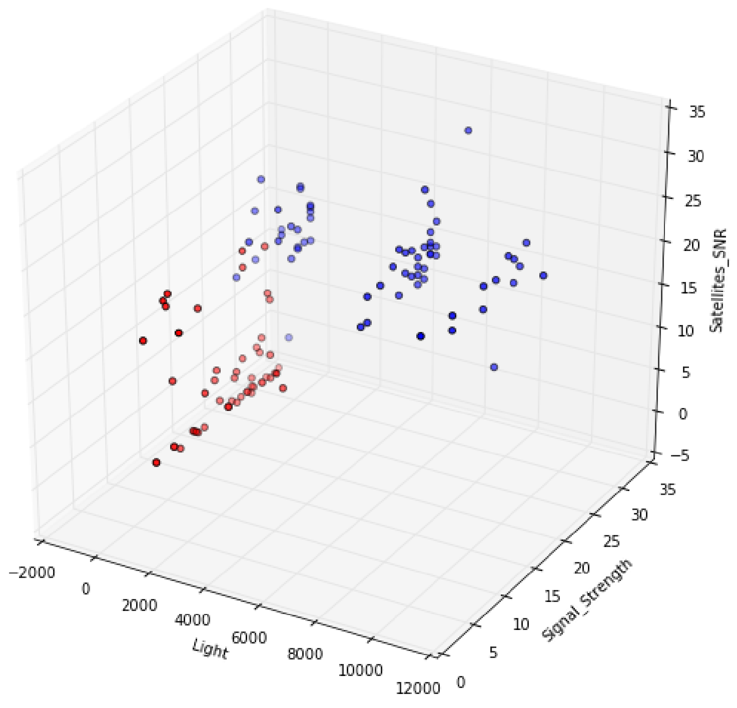


Figure 4. Training data on a 3D scatter plot

To be fully convinced we built a projection on 2D plane (Figure 5). That means that Signal Strength does not carry any valuable information for our classification.

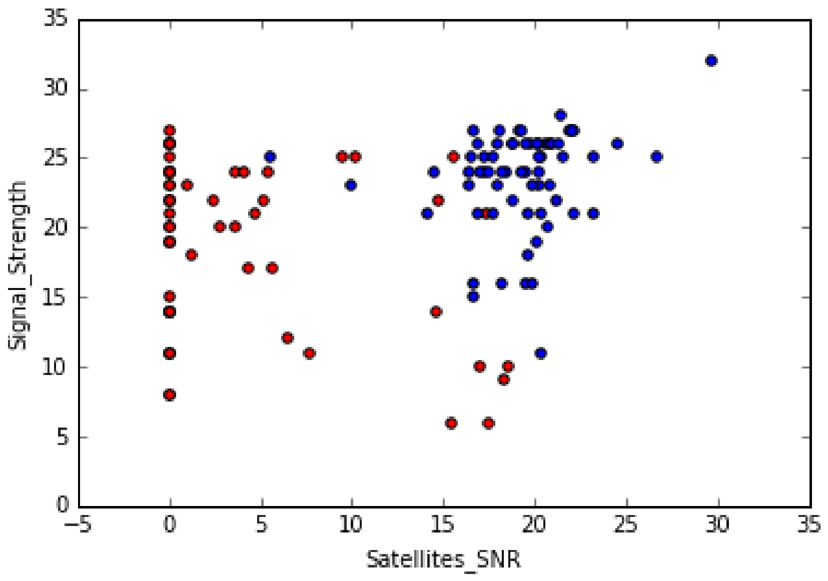


Figure 5. Projection on a 2D plane

That is why finally left only two features for each object Light and Satellites signal-to-noise ratio.

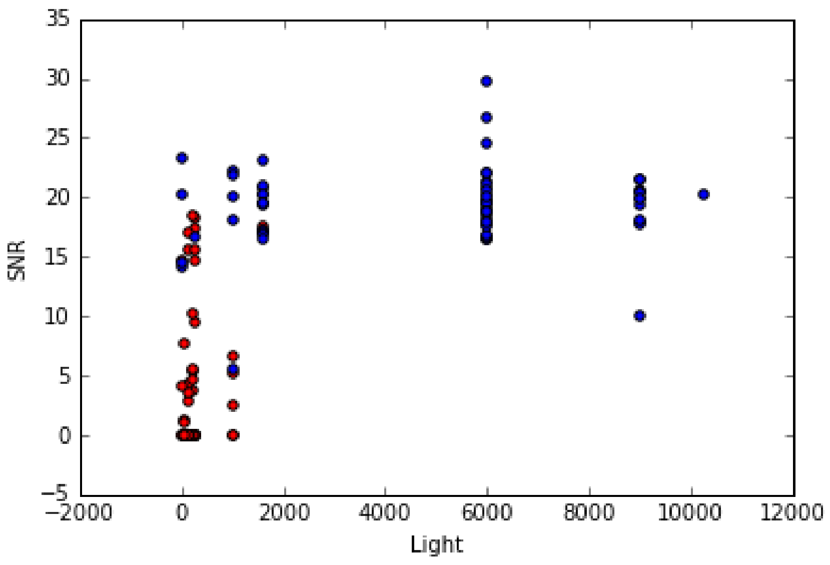


Figure 6. Resulting data on a 2D scatter plot

Finally, we had to choose between a plenty of existing linear classifiers. To resolve this issue we decided to try three popular classifiers and take the one that will show the best result on our tests. Next, we provide a description for each classifier and the results.

4.2. Decision Tree

Decision tree classifier is a supervised learning method. It learns simple decision rules from the data and uses them to predict target classes. One of the main advantages of decision tree is that it is easy to use, understand, interpret and visualize.

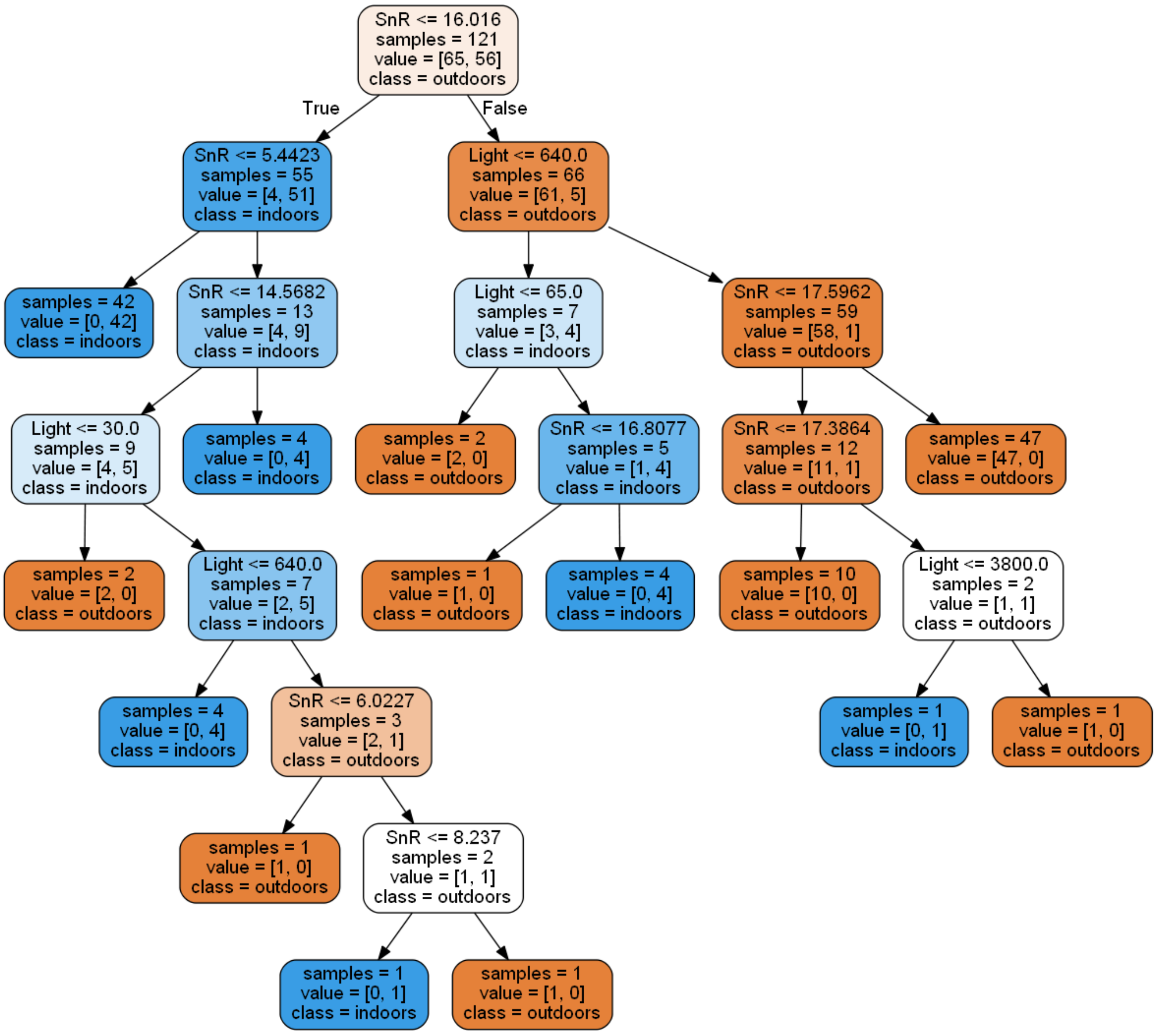
Decision tree is built in a following way. Input (or training) data is organized in the form of records $(\vec{x}, Y)$ where $\vec{x}$ is a vector of features and Y is a class label. The tree learns by splitting the data into two parts by choosing a best split in terms of a certain metric. This process is repeatead on each of the two parts until all the parts have objects of the same class only. There are many possible ways to measure the best split but we used Gini impurity metric that represents how often would a randomly chosen element from the training dataset be labeled incorrectly. For two class classification it can be calculated as follows

$$I_G = \frac{n_0}{m} * (1 - \frac{n_0}{m}) + \frac{n_1}{m} * (1 - \frac{n_1}{m})$$

where n_i is the number of items labeled i and m is the number of items in the input dataset. On the node where all objects belong to one class it reaches its minimum value(zero).

After that, trained decision tree classifier can be used to predict class of a new object described by the same type of features.

We trained a decision tree on our data and made a decision graph (Figure 7). Resulting classifier is significantly overfitted because we have many nodes with a small number of objects which means that this tree does not generalize data well. To prove this we provide a corresponding scatter plot with a decision boundary.



SnR <= 16.016 samples = 121 value = [65, 56] class = outdoors
True
False
SnR <= 5.4423 samples = 55 value = [4, 51] class = indoors
Light <= 640.0 samples = 66 value = [61, 5] class = outdoors
samples = 42 value = [0, 42] class = indoors
SnR <= 14.5682 samples = 13 value = [4, 9] class = indoors
Light <= 65.0 samples = 7 value = [3, 4] class = indoors
SnR <= 17.5962 samples = 59 value = [58, 1] class = outdoors
Light <= 30.0 samples = 9 value = [4, 5] class = indoors
samples = 4 value = [0, 4] class = indoors
samples = 2 value = [2, 0] class = outdoors
SnR <= 16.8077 samples = 5 value = [1, 4] class = indoors
SnR <= 17.3864 samples = 12 value = [11, 1] class = outdoors
samples = 47 value = [47, 0] class = outdoors
samples = 2 value = [2, 0] class = outdoors
Light <= 640.0 samples = 7 value = [2, 5] class = indoors
samples = 1 value = [1, 0] class = outdoors
samples = 4 value = [0, 4] class = indoors
samples = 10 value = [10, 0] class = outdoors
Light <= 3800.0 samples = 2 value = [1, 1] class = outdoors
samples = 4 value = [0, 4] class = indoors
SnR <= 6.0227 samples = 3 value = [2, 1] class = outdoors
samples = 1 value = [0, 1] class = indoors
samples = 1 value = [1, 0] class = outdoors
samples = 1 value = [1, 0] class = outdoors
SnR <= 8.237 samples = 2 value = [1, 1] class = outdoors
samples = 1 value = [0, 1] class = indoors
samples = 1 value = [1, 0] class = outdoors

Figure 7. Overfit decision graph

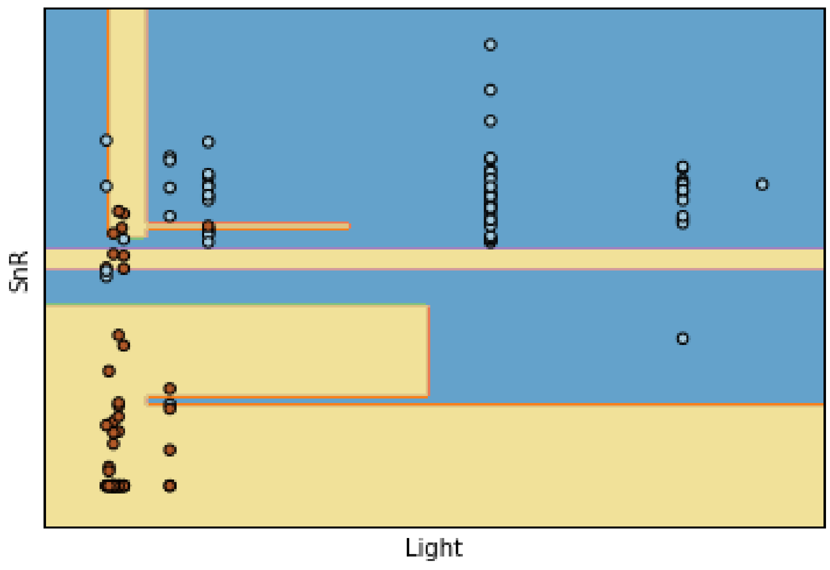


Figure 8. Overfit decision tree

To avoid this we can limit the minimum number of objects in a leaf by 5. The resulting graph is presented on Figure 9. We also built a decision boundary on scatter plot for better visualization of classification (Figure 10).

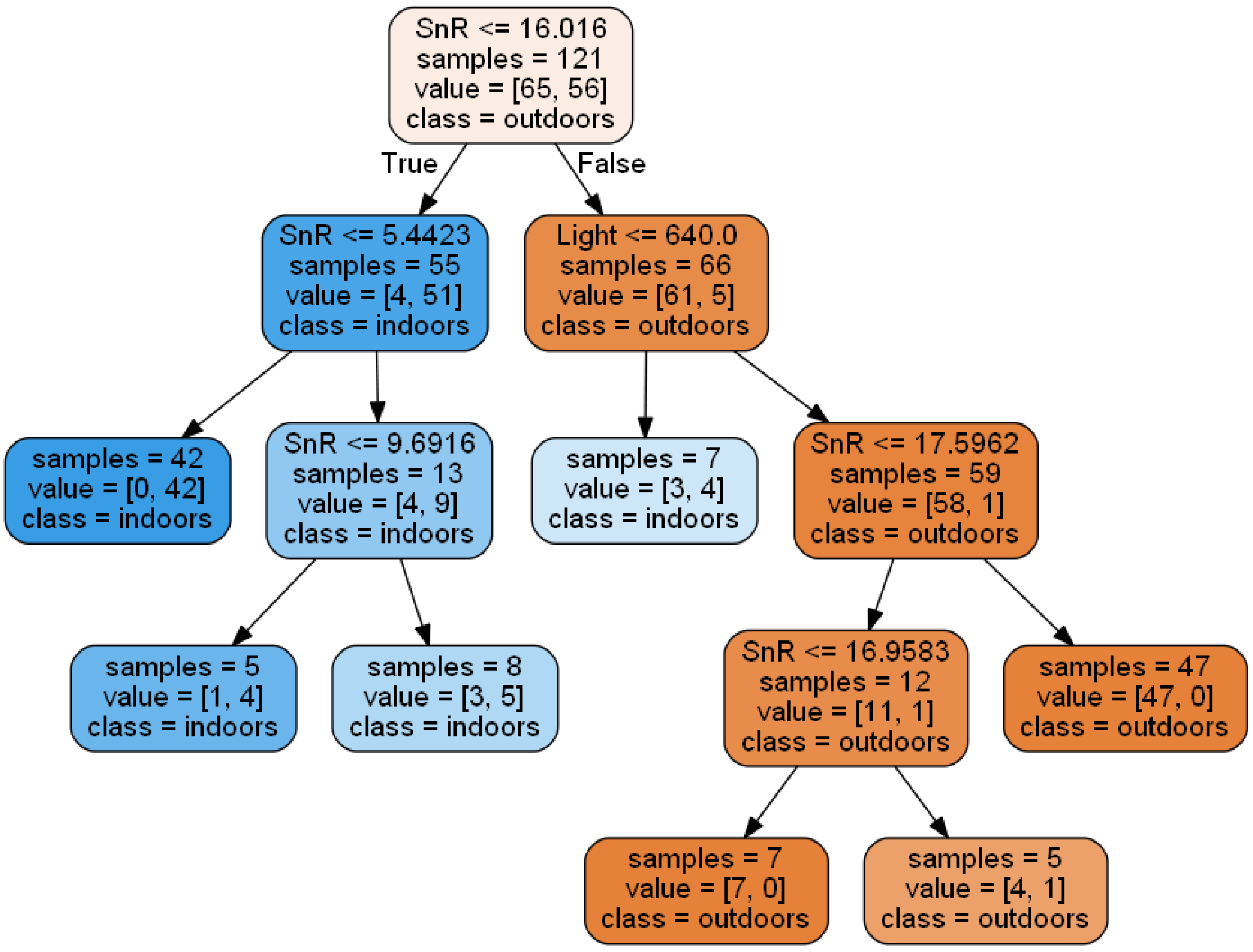


Figure 9. Pruned decision graph

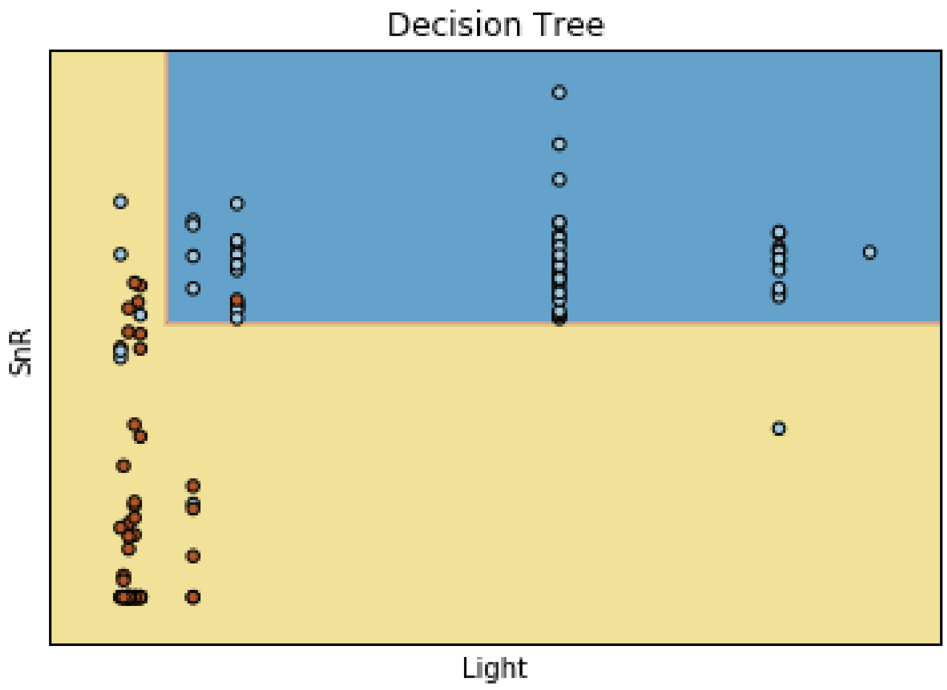


Figure 10. Scatter plot with decision boundary

4.3. Support Vector Machine with linear kernel

Support Vector Classifier is a supervised learning method. In two class case its goal is to build a division hyperplane in the feature space that will maximize the gap between classes in order to minimize the generalization error. Border hyperplanes of this gap can be written as

$$\vec{w} \cdot \vec{x} - b = 1$$

$$\vec{w} \cdot \vec{x} - b = -1$$

which means that each object $(\vec{x}_i, Y_i)$ satisfies the following inequalities

$$\vec{w} \cdot \vec{x}_i - b \geq 1, \quad \text{if} \quad Y_i = 1$$

$$\vec{w} \cdot \vec{x}_i - b \leq 1, \quad \text{if} \quad Y_i = -1$$

which is the same as

$$Y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 \quad \text{for all} \quad i$$

In order to be able to use SVM when the data is not linearly separable we should include a new variable ε and rewrite the inequality:

$$Y_i(\vec{w} \cdot \vec{x}_i - b) \geq 1 - \varepsilon_i \quad \text{for all} \quad i$$

Which brings us to the optimization problem:

$$minimize \quad \frac{1}{n} \sum_{i=1}^{m} \varepsilon_i + \lambda \|w\|^2$$

Where λ is a regularisation prameter. This problem can be solved by transforming it into a dual problem which in turn brings us to the following decision function:

$$f(x) = sgn\left(\sum_{i=1}^{n} c_i(x_i, x) + b \right)$$

Where sgn denotes the signum function, x_i are the support vectors and c_i are the coefficients that are obtained as the solution of the dual problem.

We trained an SVM classifier with linear kernel on our data and used exhaustive grid search to find the regularisation parameter λ that will show the best cross-validation score. It was $\lambda = 0.01$. Again we built the decision boundary on a scatter plot (Figure 11).

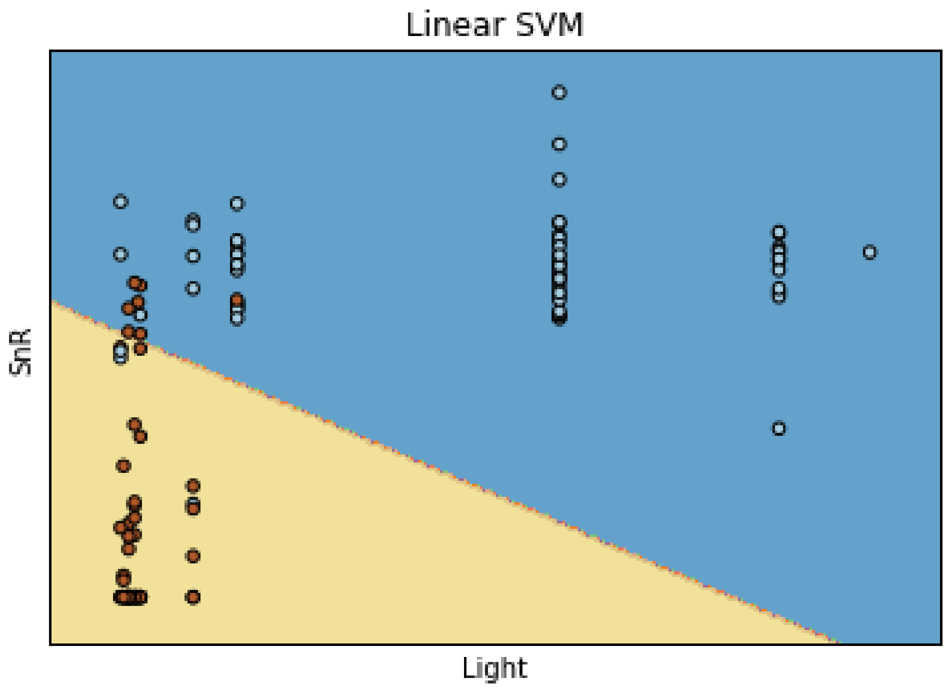


Figure 11. Scatter plot with decision boundary

4.4. Logistic regression

Logistic regression is a classifier that uses decision function that is close to SVMs:

$$f(x) = sgn\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

The difference is in the way the weights w_i are obtained:

$$\vec{w} = argmin\left(\frac{1}{2}(\vec{w},\vec{w}) + C\sum_{i=1}^{m} \ln(\exp(-y_i((\vec{w},\vec{x_i}) + b)) + 1)\right)$$

It allows to predict posterior probabilities for each class but we do not need that for our task.

A scatter plot with a decision boundary for Logistic regression is provided on Figure 12.

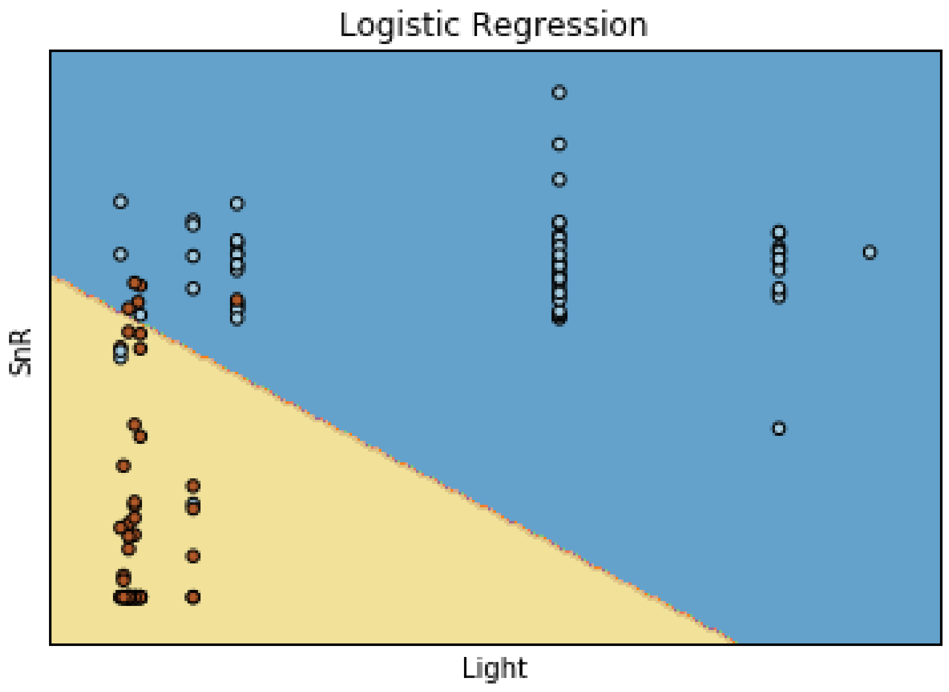


Figure 12. Scatter plot with decision boundary

4.5. Result

To decide which classifier is better, we decided to use cross-validation. The result was:

- Decision Tree: 0.885
- Linear SVM: 0.926
- Logistic regression: 0.917

So SVM was chosen and the next step was to implement it on Android.

5. Implementing classifier

To implement the decision function, we had to obtain intercept (b in the decision function), dual coefficients (c_i) and support vectors (x_i). Scikit-learn allows to do that easily so implementation in java is presented below.

```
public float sc_mul(double[] x1,
                    double[] x2){
  int result = 0;
  for(int i = 0 ; i< x1.length; i++){
        result += x1[i]*x2[i];
  }
  return result;
}

public boolean predict(double[] X,        // Input vector
                       double b,          // Intercept
                       double[] w,        // Dual coefficients
                       double[][] sv){    // Support vectors
  int n = sv.length;
  int s = 0;
  for(int i = 0 ; i<n;i++){
        s += w[i]*sc_mul(sv[i], X);
  }
  return s + b > 0;
}
```

6. Conclusion

We presented a fully functional Android application that allows to predict if the device is located inside or outside of the building and visualize measurements of some mobile sensors. We studied three different classifiers and discovered which one manages to show the best results.

Final product shows accurate predictions except some difficult cases like when the phone is next to a window which makes both light and SnR very high or, in contrast, at night and when there are some obstacles that reduce SnR that confuse the classifier.

The further work on the created application may include gathering bigger amounts of train data and providing support for other sensors such as temperature and magnetometer. Also more complex machine learning methods may be used.

References

[1] Trevor Hastie, Robert Tibshirani and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.*. Springer, 2009.

[2] Android reference,
`https://developer.android.com/reference/packages.html`

[3] Scikit-learn reference,
`http://scikit-learn.org/stable/documentation.html`