Automated collision detection using machine learning algorithms on sensor data

Project Report

from the Course of Studies Angewandte Informatik

at the Cooperative State University Baden-Württemberg Mannheim

by

Tim Schmidt

August 2018

Time of Project	17 weeks
Student ID, Course	8531806, TINF15AI-BC
Company	IBM Deutschland GmbH, Ehningen
Supervisor in the Company	Julian Jung
Reviewer	Julian Jung

Author's declaration

Hereby I solemnly declare:

1. that this Project Report, titled *Automated collision detection using machine learning algorithms on sensor data* is entirely the product of my own scholarly work, unless otherwise indicated in the text or references, or acknowledged below;
2. I have indicated the thoughts adopted directly or indirectly from other sources at the appropriate places within the document;
3. this Project Report has not been submitted either in whole or part, for a degree at this or any other university or institution;
4. I have not published this Project Report in the past;
5. the printed version is equivalent to the submitted electronic one.

I am aware that a dishonest declaration will entail legal consequences.

Mannheim, August 2018

Tim Schmidt

Contents

List of Figures

1 Introduction

1.1 Motivation

The field of data analytics and machine learning is increasingly becoming the main differentiator of industry competition. This is especially true in combination with another growing field, the Internet of Things (IoT), the McKinsey Global Institute's study 'The Age of Analytics: Competing In A Data-Driven World' found. [1] IBM has set a goal to tackle the challenges of IoT and Big Data with an array of new services, industry offerings and capabilities for enterprise clients, startups and developers. [2] These services are part of IBM Bluemix, a cloud plaform as a service (PaaS).

An important step for IBM is getting clients interested and aware of the capabilities of IoT and machine learning so that collaborations and projects with clients can be established. One way this is done is by presenting showcases – small projects that show in an exemplary way the possibilities and values of the technology. It is essential to show technologies from the IBM Bluemix portfolio to present their capabilities and outline differentiating factors to competitors.

The Sensorboard is such a showcase. A physical skateboard is mounted with a Texas Instrument's SensorTag, and is able to connect wirelessly to the cloud platform IBM Bluemix. From there one or more Sensorboards can be monitored and managed. As part of the showcase a the Sensorboard is demonstrated as a rentable and cloud-managed vehicle for urban mobility.

This showcase will be extended with machine learning applications to present the value machine learning algorithms can have on sensor data. The specific goal of that extension is to recognize collision that happen to the Sensorboard and that indicate accidents. This recognition will be based solely on sensor data and patterns in the time series data. For this IBM Bluemix services are used to showcase the capabilities of the platform. The services which are requested to be demonstrated and used in the showcase are the IBM IoT Foundation, to connect the Sensorboard to the platform, and the IBM Data Science Experience as a development environment for an underlying Apache Spark cluster, on which machine learning models will be created.

1.2 Objective

A consumer grade skateboard is equipped with a Texas Instrument's SensorTag. The Texas Instrument's SensorTag collects acceleration data from the skateboard on three spacial axis. To process this data cloud services of IBM Bluemix are used. This especially includes the IBM Data Science Experience and an Apache Spark service. Consequently utilizing Apache Spark as a framework for cluster computing, machine learning models are trained using Apache Spark. This narrows down the selection of machine learning algorithms to the algorithms available in function library Apache Spark MLlib, which provides scalable algorithms, optimized to run on Apache Spark.

This paper investigates on the feasibility of machine learning algorithms for collision detection in the above described setting and identifies, if feasible, the best performing algorithm under the described conditions.

The scenario of impact detection that may indicate accidents requires a higher focus on detecting an accident than on preventing false alarms. A metric is found to represents this unequal relationship in the assessment of algorithm performance.

1.3 Limitations of the Research

The technologies described in the chapters Motivation and Objective like the Texas Instrument's SensorTag, the IBM Bluemix services and programming libraries are a given for the project by IBM. Therefore reasons for this particular decisions will not be discussed and no evaluation of alternatives will be conducted in this paper. The findings of this paper are acquired and only applicable for the specific hardware and configuration described in the paper. Findings can not be transferred to other vehicles, sensors, configurations or any other setup than the one described in this paper. The data which is used to come to decisions and to train models is generated in experiments, which are conducted solely for this reason. No other sensor data of skateboards or other vehicles are used in the considerations of this paper. The machine learning algorithms examined only include algorithms currently provided by the function library Apache Spark MLlib. A list of relevant algorithms currently provided can be seen in the appendix in chapter ??.

1.4 Organization of the Research

In the beginning the chapter 'Theory' introduces the topic of machine learning. Relevant machine learning algorithms and evaluators are explained with sufficient theoretical and mathematical background information. The goal of the chapter is to create a basic understanding for methods and metrics this paper investigates.

Following, in the chapter ’Analysis of the pattern recognition problem’, the pattern recognition problem will be analysed in detail. The goal of the chapter is to define specific possible solutions for the problem.

In the chapter ’Algorithm configuration’, bringing together the theoretical knowledge about machine learning and possible solution strategies, it is shown how algorithms are best configured and applied for the problem. The goal of the chapter is to generate a comparable result for the different possible solutions.

In the chapter ’Evaluation’ the results are then compared to each other. The goal of this chapter is to answer the question of whether machine learning algorithms are feasible for collision detection. If feasible a winning solution that is best performing under the considered conditions is identified. A conclusions about the results is drawn.

In the end, the chapter ’Outlook’ gives a proposal on possible further research and development.

2 Theory

2.1 Machine Learning

2.2 Classification

2.3 Evaluation of Classifiers

- f-score
- k-fold validation

2.4 Algorithms

- hyper-parameters

2.4.1 Linear support vector machine

2.4.2 Logistic regression

2.4.3 Decision tree

2.4.4 Random forest

2.4.5 Gradient-boosted trees

2.4.6 Naive Bayes

3 Analysis of the pattern recognition problem

3.1 Analysis of the time series data

This chapter will provide a closer examination of the time series data measured by the Texas Intrument's SensorTag. This data is prerequisite for the further work of this paper. The sensor data is measured and collected at a sampling rate of 10 measurements per second. To optimize accuracy the maximal sampeling rate of the Texas Instrument's SensorTag was chosen. The measurements include acceleration in 3 spacial axis. The axis each correspond to a direction on the Sensorboard and are orthogonal to each other as shown in figure 3.1. The acceleration in any particular direction will be referred to as x, y or z acceleration.

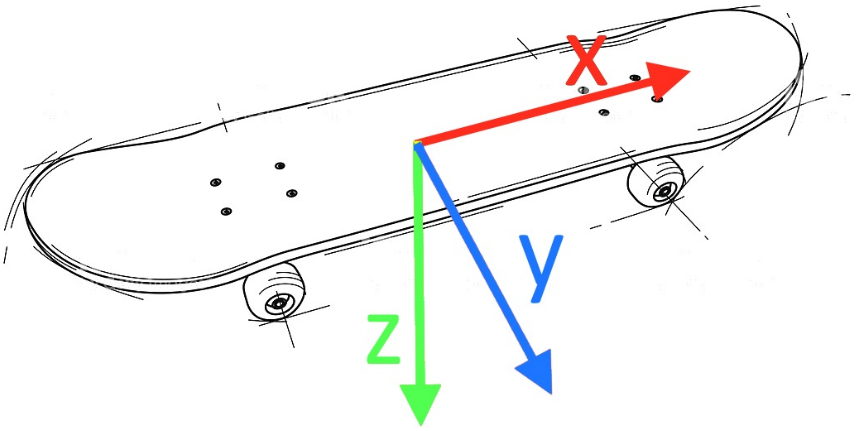


Figure 3.1: Illustration of Sensorboard and axis of measurement. The driving direction is to the right.

Those measurements are taken simultaneously and handled from then on together as one measurement collection. This collection is enriched with the time of the measurement in form of a Unix timestamp.

A measurement collection as described above will also be referred to as a data point. A single measurements (for instance x acceleration) of a data point will be also referred to as features of that data point.

An exemplary data point in a key-value-pair format can be seen here:

```
{
  "accx": -0.079590,
  "accy": 0.001465,
  "accz": -1.016602,
  "time": 1502735960911
}
```

The keys stand for acceleration in x direction, acceleration in y direction, acceleration in z direction and the Unix timestamp when the measurement took place.

Using the time of measurement, data can be plotted on a time axis plot as seen in figure 3.2.

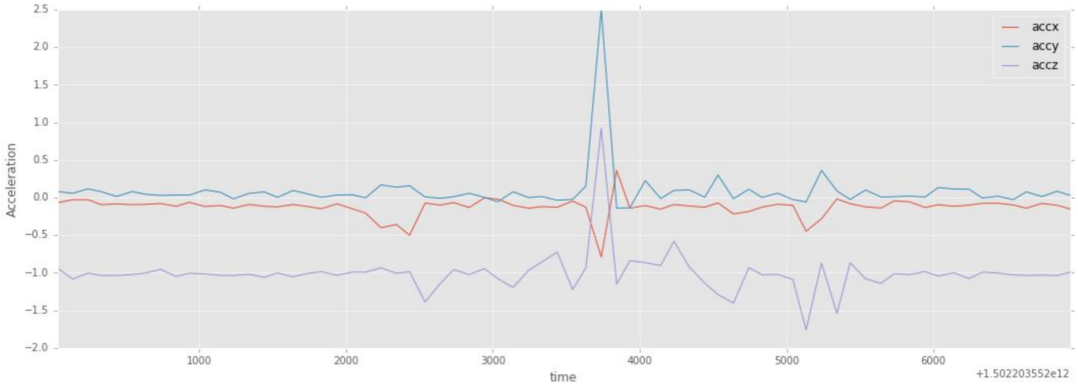


Figure 3.2: Example of measured accelerations on 3 Axis during a collision

With the Sensorboard data has been collected. During the testing of the Sensorboard 69 collisions were simulated. Accordingly 69 of the 6453 datapoints collected, are labeled as a collision while the rest is labeled as no collision. The labeling was figured out by analysing video footage recorded during the test drive. The exact time of a collision in the video was matched with the time of the recorded sensor data to label data points correctly.

The distribution of the data points labeled as collision or no collision can be seen in figure 3.3. The scatter plot shows the relationship between the x and z acceleration and the label of the data point. A blue data point represent a collision the red data points represent no collision.

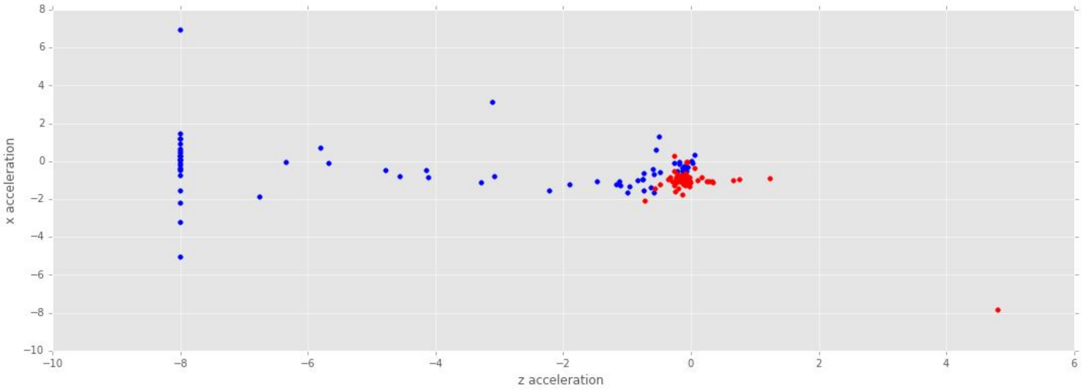


Figure 3.3: Distribution of data points labeled as collision (blue) or no collision (red)

3.2 Solution strategy

Combining the knowledge about the sensor data from the previous section and the objective as described in chapter 1, this chapter will narrow down on a concrete problem type and solution strategy.
In the process limitations described in Chapter 1 have to be considered.
Having a set of prelabeled sensor data allows for a supervised machine learning approach. Furthermore the existence of concrete data points which represent the state of the Sensorboard at a given time makes it possible to base the recognition of collisions soly on data points. The recognition of collisions can then in turn be formulated as the problem of deciding weather a given data point corresponds to a collision or no collision. A problem of this kind can be solved by binary classification.
Having specified the solution method to a binary classification approach, concrete algorithms can be considered. In chapter 1 the necessity of using the Apache Spark library module 'MLLib' is constituted. For the specific task of supervised binary classification the following algorithms are supported by the MLLib module:

- Linear SVM
- Logistic regression
- Decision tree
- Random forest
- Gradient-boosted trees
- Naive Bayes

Only the possible algorithms mentioned here will be parameterized and trained on the collected data in chapter 4.

To assess the performance of resulting models a clear metric will be used for comparison. This metric has to represent the unequal importance of detecting a collision and preventing false alarms. As mentioned in chapter 1 the act of detecting a collision that actually happened (true positives) is more important than preventing the detection of a collision that didn't happen (false positives). In a classification setting the effectiveness in detecting collision that actually happened is described by recall. The effectiveness of preventing the detection of collision that didn't happen is described by precision. The metric that will be used to represent the unequal relationship between precision and recall is a f_2-score. This metric puts more weight on the recall of a model. This way a later evaluation of trained classifiers can be done based on one correctly weighted metric.

4 Algorithm configuration

Many of the methods proposed in chapter 3 require hyper-parameters that vary aspects of the algorithm and might lead to better or worse results. To ensure that all the algorithms are comparable with each other this chapter goes over the process of finding the optimal hyper-parameters for every approach. A comparable score is assigned to every algorithm under the condition of optimal hyper-parameters.

Hyper-parameter tuning

The process of searching and finding good hyper-parameters for a given algorithm is called hyper-parameter tuning. Searching for good hyper parameters takes the following aspects to be considered:

- **an estimator**
  The estimator in this case is a binary classifier. The 8 different binary classifiers have to be considered as mentioned in chapter 3.
- **a parameter space**
  The parameter space defines a set of parameters or combination of parameters that will be tested. From this set one element will be chosen as most fitting. Because every algorithm requires different parameters, the parameter space is different for every algorithm. The parameter space will be defined in the sections below for every algorithm.
- **a method for searching or sampling candidates**
  The method that is chosen is grid search. Grid search is a brute force search, which uses an exhaustive searching through a defined parameter space. Models and iteratively trained and evaluated by a score function. The model scores are then compared. The hyper-parameters that produced the best performing model will be chosen as optimal ones for the algorithm.
- **a cross-validation scheme**
  The cross-validation scheme used is a k-fold validation, with k=10.

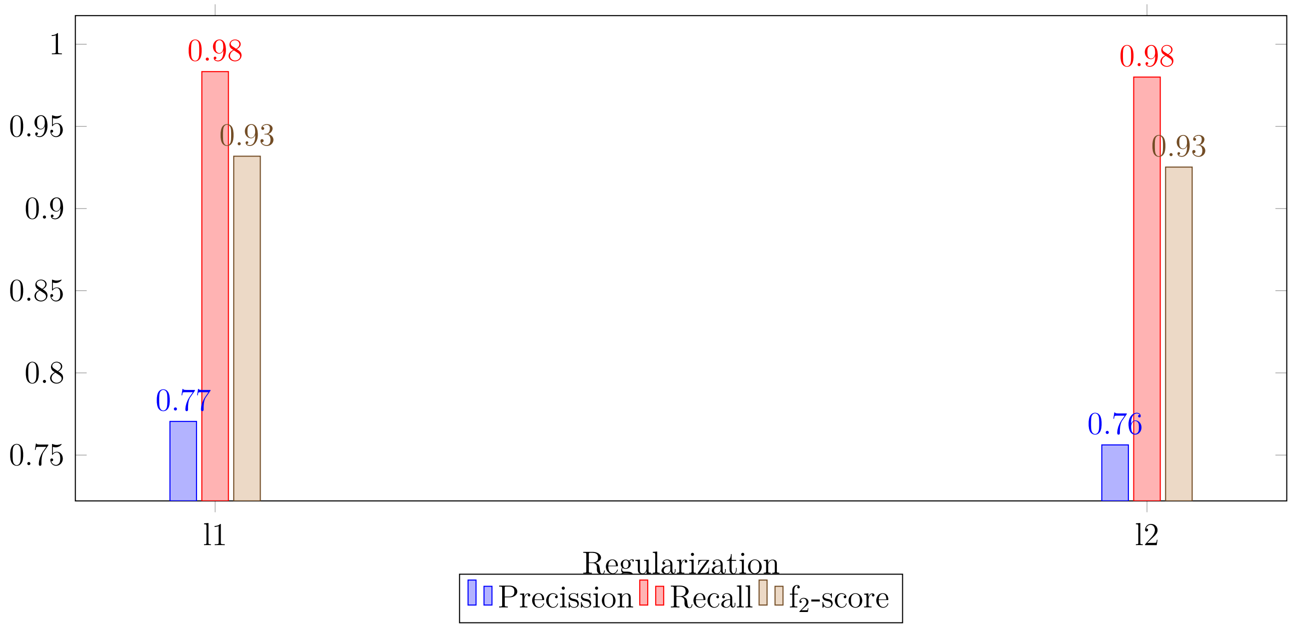
- **a score function**
  As mentioned in chapter 3 a f_2-score will be used to evaluate the models in the end. Our goal for the hyper-parameter tuning is thereby also to maximise the f_2-score. For that reason the f_2-score is used as a score function.

For the hyper-parameter search no external functions or libraries are used. The search algorithm is implemented in the following way.
For a given estimator (in this case one of the possible classification algorithms) the parameter space is defined manually for every hyper-parameter of a given estimator.
Models are now repeatedly trained by iterating through every combination of hyper-parameters in the parameter spaces.
For every trained model a k-fold validation, with k=10, is performed.
During the k-fold validation, the f_2-score is calculated.
After this process is done the hyper-parameters that resulted in the highest f_2-score are chosen.
For the each of the relevant algorithms a parameter space will be defined. The results of the hyper parameter search will then be presented.

Linear SVM

The tunable hyper-parameters for the the linear SVM classifier is the regularization.
L1 regularization:
$\sum_{i=1}^{n} |y_\mathrm{i} - f(x_\mathrm{i})|$
y_i is the target value and x_i is the estimated value for y_i.
L1 regularization is a loss function that minimizes the sum of the absolute differences of target and estimated value.
L2 regularization:
$\sum_{i=1}^{n} (y_\mathrm{i} - f(x_\mathrm{i}))^2$
y_i is the target value and x_i is the estimated value for y_i.
L2 regularization is a loss function that minimizes the square of the absolute differences of target and estimated value.

The hyper-parameter search reveals the following results for the two regularization approaches:



The best performing SVM model reaches a maximal f_2-score of **0.93** (precision: 0.77, recall: 0.98) with the following hyper-parameter:

Regularization: **l1 regularization**

Logistic regression

The tunable hyper-parameters for the the logistic regression classifier is the regularization.

L1 regularization:

$\sum_{i=1}^{n} |y_\mathrm{i} - f(x_\mathrm{i})|$

$\mathrm{y_i}$ is the target value and $\mathrm{x_i}$ is the estimated value for $\mathrm{y_i}$.

L1 regularization is a loss function that minimizes the sum of the absolute differences of target and estimated value.

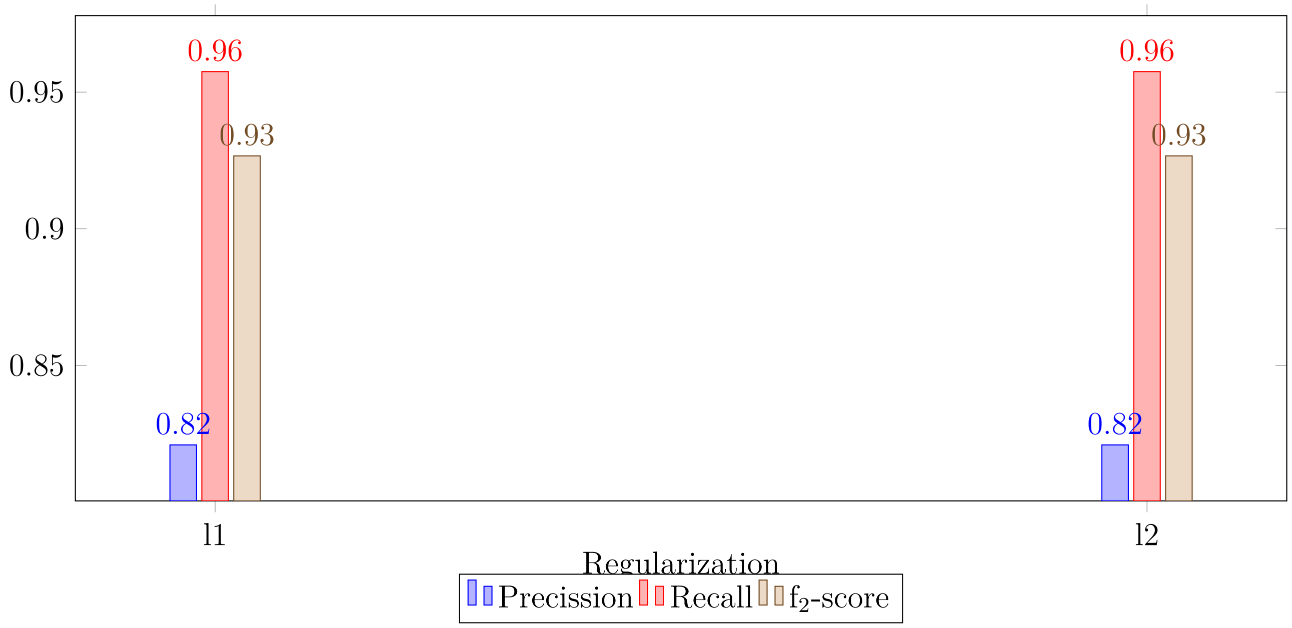
L2 regularization:

$\sum_{i=1}^{n} (y_\mathrm{i} - f(x_\mathrm{i}))^2$

$\mathrm{y_i}$ is the target value and $\mathrm{x_i}$ is the estimated value for $\mathrm{y_i}$.

L2 regularization is a loss function that minimizes the square of the absolute differences of target and estimated value.

The hyper-parameter search reveals the following results for the two regularization approaches:



The best performing logistic regression model reaches a maximal f_2-score of **0.93** (precision: 0.82, recall:0.95) independent of which regularization is used.

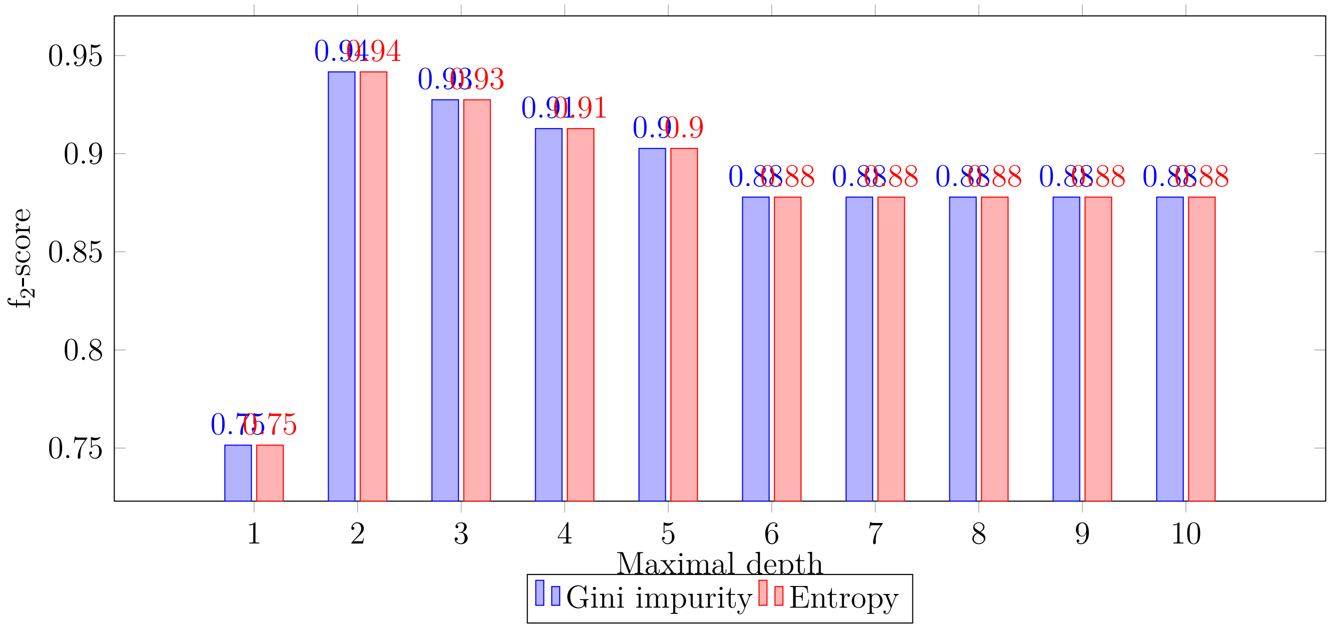
Decision tree

The tunable hyper-parameters for the decision tree algorithm are:

- **Impurity**
  The impurity measure is used to decide on a splitting rule for each node in the decision tree building process. Two impurity measures for classification are provided:
  Gini impurity:
  $\sum_{i=1}^{C} f_{\mathrm{i}}(1 - f_{\mathrm{i}})$
  C is the number of unique labels and f_{i} is the frequency of label i in a node.
  Entropy:
  $\frac{1}{N} \sum_{i=1}^{C} -f_{\mathrm{i}} \log(f_{\mathrm{i}})$
  C is the number of unique labels and f_{i} is the frequency of label i in a node.

- **Maximal depth**
  The maximal depth determines a limit of growth for a decision tree. This consequentially limits the number of nodes or decisions that are possible. It represents a stopping criteria.
  The parameter space for the maximal depth is chosen **from 1 to 10** (inclusive).

The hyper-parameter search reveals the following results:



The best performing decision tree model reaches a maximal f$_2$-score of **0.94** (precision: 0.92, recall: 0.96) with the following hyper-parameters:

Impurity:	**Ginni Impurity** or **Entropy**
Maximal depth:	**2**

Random forest

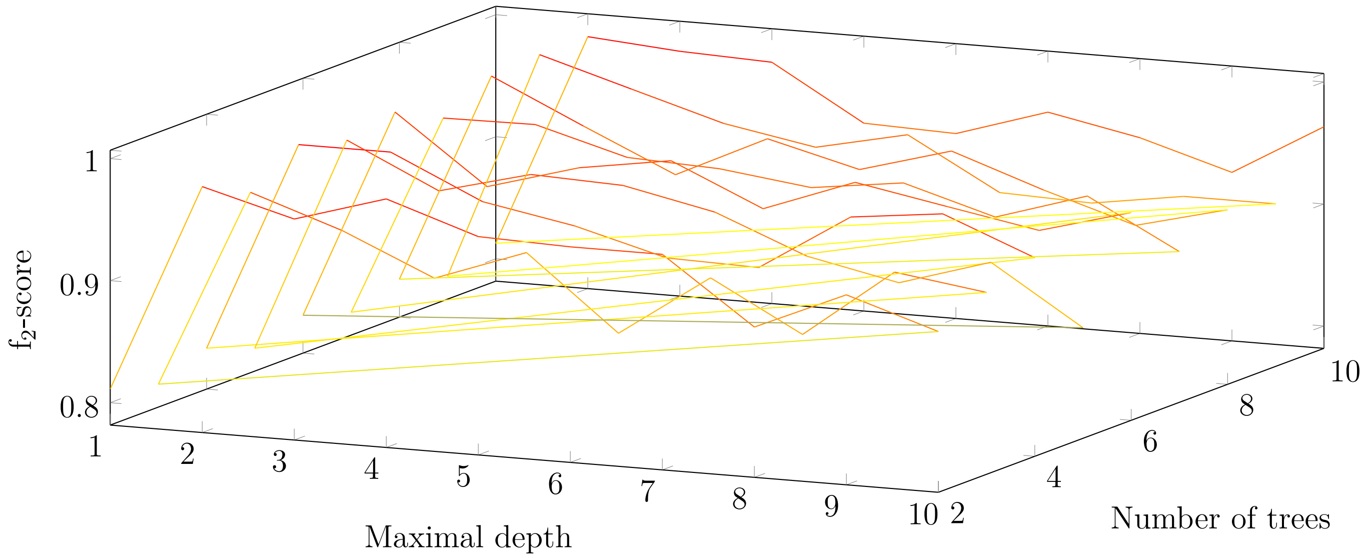
The tunable hyper-parameters for the random forest classifier are:

- **Number of trees**
  The number of trees that are build during the training phase to act as an ensemble of (randomized) decision trees.
  The parameter space for the number of threes is chosen **from 2 to 10** (inclusive).

- **Impurity**
  The impurity measure is used to decide on a splitting rule for each node in the decision tree building process. Two impurity measures for classification are provided:
  Gini impurity:
  $\sum_{i=1}^{C} f_{\mathrm{i}}(1 - f_{\mathrm{i}})$
  C is the number of unique labels and f_i is the frequency of label i in a node.
  Entropy:
  $\frac{1}{N}\sum_{i=1}^{C} -f_{\mathrm{i}} \log(f_{\mathrm{i}})$
  C is the number of unique labels and f_i is the frequency of label i in a node.

- **Maximal depth**
  The maximal depth determines a limit of growth for a decision tree. This consequentially limits the number of nodes or decisions that are possible. It represents a stopping criteria.
  The parameter space for the maximal depth is chosen **from 1 to 10** (inclusive).

The hyper-parameter search reveals the following results:



The best performing decision tree model reaches a maximal f_2-score of **0.99** (precision: 0.94, recall: 1.0) with the following hyper-parameters:

Number of trees:	**4**
Impurity:	**Entropy**
Maximal depth:	**2** or **3**

or

Number of trees:	**9** or **10**
Impurity:	**Entropy**
Maximal depth:	**2**

Gradient-boosted trees

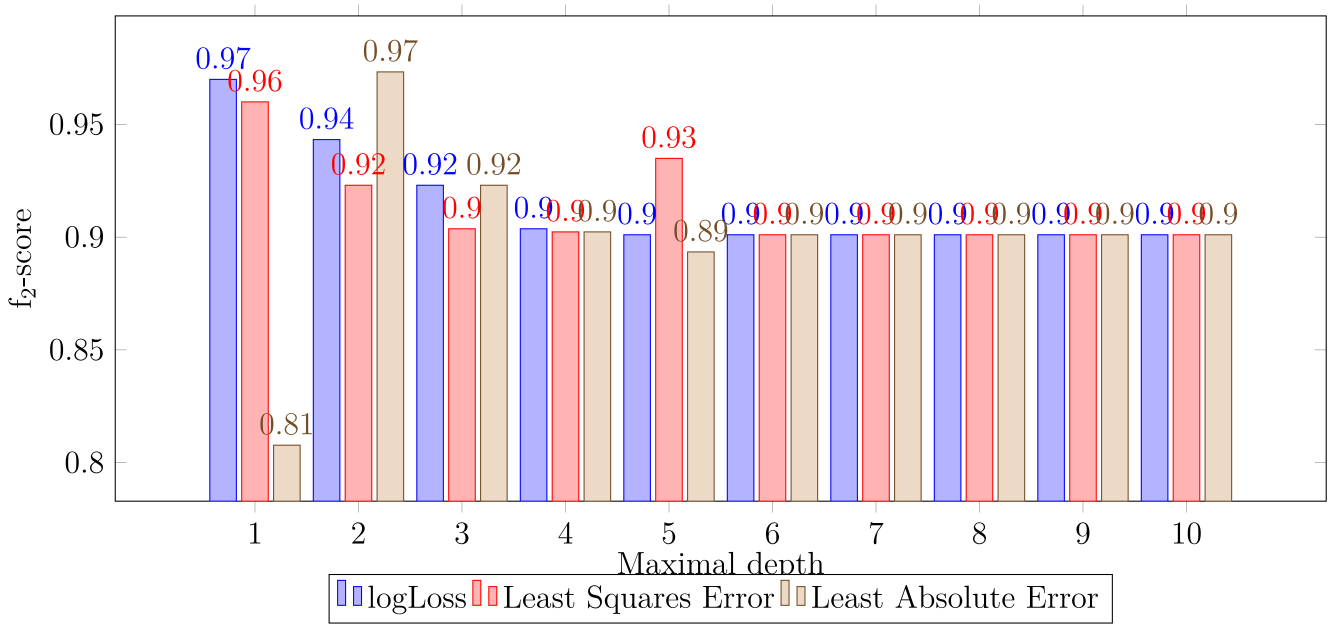
The tunable hyper-parameters for the decision tree algorithm are:

- **Loss function**
  The loss function is used for minimization during gradient boosting. The loss function provided are:
  LogLoss
  Least Squares Error

Least Absolute Error

- **Maximal depth**
  The maximal depth determines a limit of growth for a decision tree. This consequentially limits the number of nodes or decisions that are possible. It represents a stopping criteria.
  The parameter space for the maximal depth is chosen **from 1 to 10** (inclusive).

The hyper-parameter search reveals the following results:



The best performing Gradient-boosted tree classifier reaches a maximal f_2-score of **0.97** (precision: 0.96, recall: 0.98) with the following hyper-parameters:

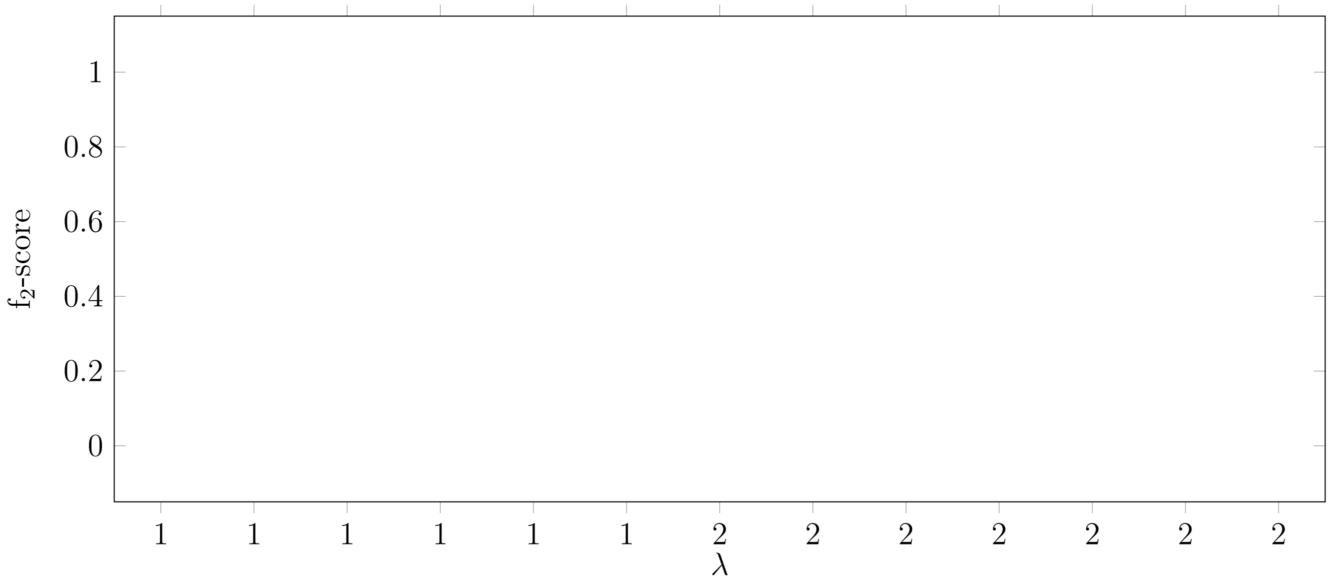
Loss function: **Least Absolute Error**

Maximal depth: **2**

Naive Bayes

The tunable hyper-parameters for the the Naive Bayes classifier is a λ -value used for additive smoothing. The parameter space for λ is chosen **from 0.1 to 10.0** (inclusive).

The hyper-parameter search reveals the following results for the two regularization approaches:



The best performing naive bayes classifier reaches a maximal f$_2$-score of (precision: , recall:) with the hyper-parameter:

λ :

5 Evalutation

The optimal configuration for the different algorithms was worked out. Now this chapter is devoted to comparing and deciding on an optimal algorithm for the problem of collision detection.
To be able to compare the algorithms they are each used to train a model with the hyper-parameters found in chapter 3. The models are evaluated with a k-fold validation, with k=10. The results can be seen in figure 5.1.

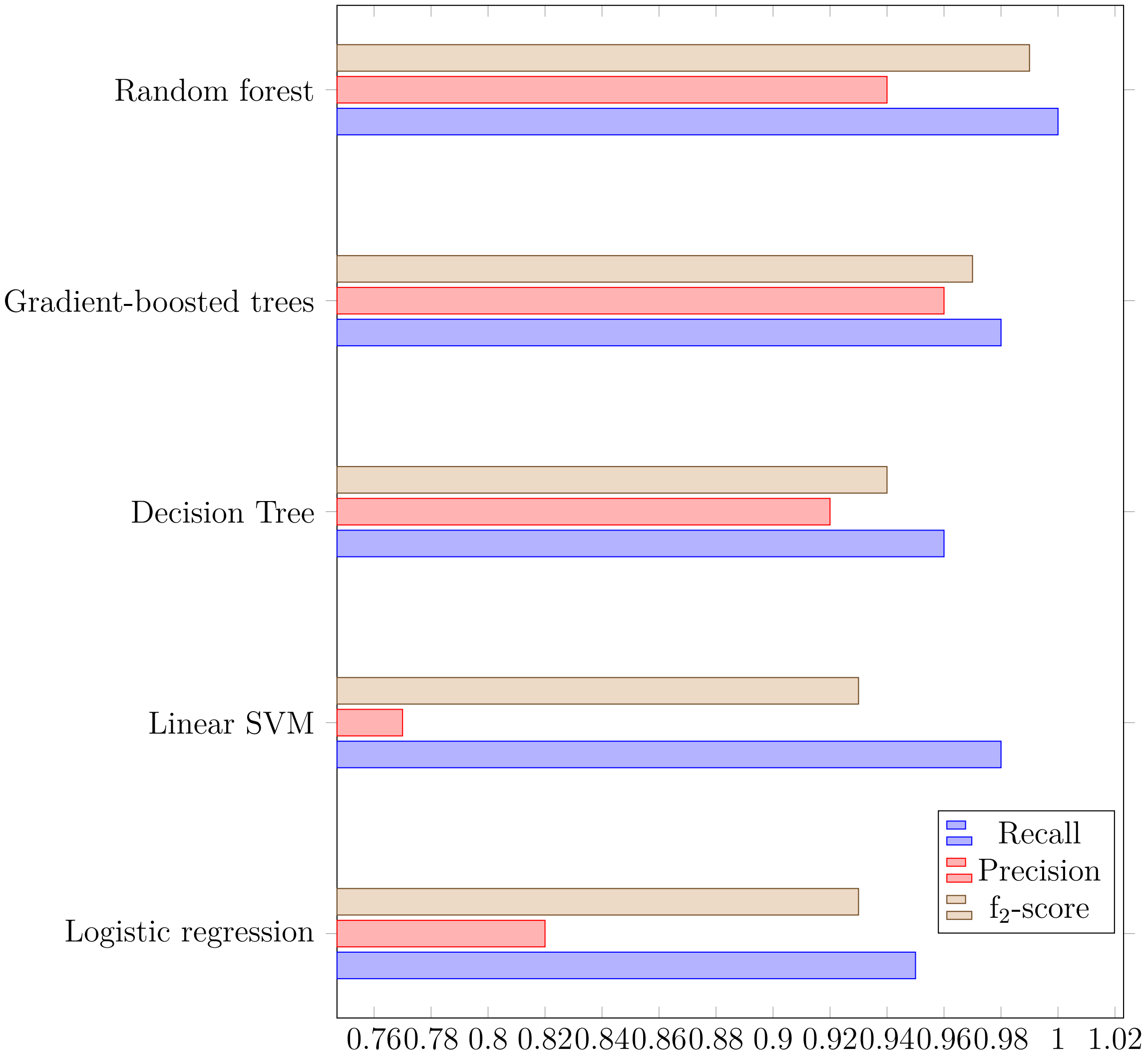


Figure 5.1: Ranking of classification algorithms.

According to the f_2-score the best performing method is a random forest classifier with 4 equally well performing hyper-parameter configurations:

Number of trees: **4**	Impurity: **Entropy**	Maximal depth: **2**
Number of trees: **4**	Impurity: **Entropy**	Maximal depth: **3**
Number of trees: **9**	Impurity: **Entropy**	Maximal depth: **2**
Number of trees: **10**	Impurity: **Entropy**	Maximal depth: **2**

The best performing random forest model reaches a maximal f_2-score of **0.99** (precision: 0.94, recall: 1.0).

This results indicate that the machine learning approach to recognizing collisions on a physical board is eligible and can produce good performance. Still it has to be considered that these results are achieved on data which is generated in experience soly for that reason and results may differ when applied in real world situations.

The resulting scores for the different classifiers reveal further insights on the task of this usecase. In figure 5.1 the ranking of the classification methods shows that decision tree based approaches have a advantage over the remaining algorithms. That suggests that non-linear models perform better on the given data set than linear models. In turn it can be said that the non-linear nature of the dataset favors the use of non-linear classifiers like decision trees or assembles of decision trees.

Random forest and gradient-boosted tree models as decision tree assembles performed on average better than a single decision tree. Assembles methods reduce the prediction variance of a classifier and thereby prevent overfitting and generalize better. The sparse dataset leads a decision tree classifier to slight overfitting which is reduced by a decision tree assemble. Therefore the random forest and gradient-boosted tree models might perform better on the given task.

6 Outlook

As seen in chapter **??** Models work good on testdata.
In the future: Apply models on REAL sensordata, not lab environment generated Testdata!
Transpose learned knowledge and approaches to other similar problems like cars, motorbikes, other vehicles.

Bibliography

[1] R.S. Michalski,J.G. Carbonell,T.M. Mitchell. Machine Learning: An Artificial Intelligence Approach. https://books.google.de/books?hl=de&lr=&id=-eqpCAAAQBAJ&oi=fnd&pg=PA2&dq=study+on+potential+of+machine+learning&ots=Wl-OMF6Ji-&sig=d1L-cfXoFQlv0TbHk1pj18Q6rsA#v=onepage&q=study%20on%20potential%20of%20machine%20learning&f=false, Zugriff 20.08.2017