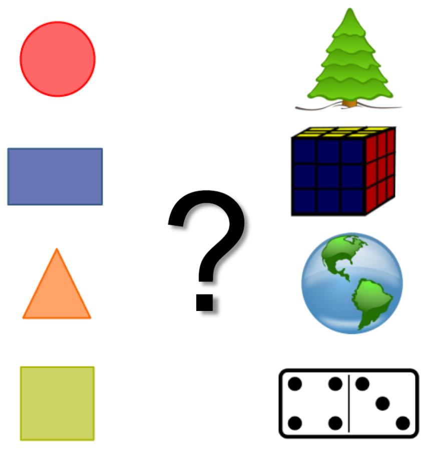
CHALMERS
UNIVERSITY OF TECHNOLOGY



Matching Vehicle Sensors to Reference Sensors using Machine Learning Methods

Object matching by supervised learning of extracted features from radar and camera sensors to LIDAR reference system.

Master's thesis in Systems, Control and Mechatronics

David Sondell
Kim Svensson

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Master's thesis 2018:EX062

Matching Vehicle Sensors to Reference Sensors using Machine Learning Methods

Object matching by supervised learning of extracted features from radar and camera sensors to LIDAR reference system.

David Sondell

Kim Svensson

Department of Electical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2018

Matching Vehicle Sensors to Reference Sensors using Machine Learning Methods
DAVID SONDELL
KIM SVENSSON

Advisor: Mohammad Hossein Moghaddam, Department of Electrical Engineering
Supervisor: Amar Shati, Volvo Car Corporation
Examiner: Henk Wymeersch, Department of Electrical Engineering

Master's Thesis 2018:EX062
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Object matching visualization indicating matches between primitive shapes as convex hull and the objects matched.

Typeset in LaTeX
Gothenburg, Sweden 2018

Matching Vehicle Sensors to Reference Sensors using Machine Learning Methods
David Sondell
Kim Svensson

Department of Electrical Engineering
Chalmers University of Technology

Abstract

Vehicle manufacturers equip their products with many sensors which are used to analyze the surroundings. In this thesis the environment is described with a set of high level objects where an object can be either a vehicle, a person or an animal. These objects are given different properties such as position and heading relative to the ego-vehicle.

It is of interest to know the accuracy of the measurements describing the high level objects. To measure the accuracy one must first find the true state of the high level objects. In this work measurements from a more accurate system that tracks the same high level objects have been considered the ground truth.

A matching algorithm was developed which matches the high level objects to ground truth objects. The matching was done using a variety of machine learning algorithms where some reach a correct classification rate of $\sim 99\%$ with a low sensitivity to added noise.

Keywords: Feature extraction, object matching, machine learning, lidar, radar, sensor fusion

Acknowledgements

We would like to thank Amar Shati for going far beyond what is expected of a supervisor especially in terms of explaining system setups, providing experimental knowledge and guiding us through gigabytes of log data. We would also like to thank Elías Marel at Volvo for providing insight in the early stages of the project and for the work he did on finding additional annotated data.

David Sondell & Kim Svensson, Gothenburg, May 2018

Contents

List of Figures

List of Tables

List of Abbreviations

Volvo	Volvo Car Corporation
AD	Autonomous driving
ML	Machine Learning
ANN	Artificial Neural Network
ReLU	Rectified Linear Units
SVM	Support Vector Machine
QP	Quadratic Programming
kNN	k-nearest neighbors algorithm
PCA	Principal Component Analysis
kPCA	Kernel Principal Component Analysis

List of Symbols

X	Data matrix, *rows* **x** *columns.*
m	Number of rows in X.
n	Number of columns in X.
$\widetilde{(\cdot)}$	Centered matrix.
I_m	Identity matrix of size m.
1_m	Square matrix of size m, where all entries are equal to 1.
λ	Eigenvalues.
$\mathcal{L}(\dots)$	Lagrange multiplier.
$\Phi,\ \phi(.)$	Nonlinear transformation.
$\mathcal{F}$	Feature space.
$\mathbb{R}^N$	Real space in N:th dimension.
$\kappa(\dots)$	Kernel function.
$\forall$	Universal quantifier meaning, *given any.*
$\in$	Set membership.
$\lvert\mathcal{S}\rvert$	Cardinality of the set $\mathcal{S}$.
$\lvert\lvert\mathbf{V}\rvert\rvert$	Norm of the vector $\mathbf{V}$.
NaN	Not a Number, missing data-sample
$\odot$	Hadamard product, element-wise multiplication

1 Introduction

This chapter introduces the thesis as a whole, starting with some background to the problem. In the aim section the goal of the project is presented followed by a scope and boundaries section which states what has not been included in the thesis. A more general outline of the rest of the thesis is presented at the end of this chapter.

1.1 Background

As advancements in automated vehicles are made, more tasks previously performed by the human driver can be taken over by the vehicle. The increase of tasks performed by the vehicle push towards the launch of fully autonomous vehicles which is predicted to have an extensive effect on traffic-related casualties as a majority of those are caused by human error [1], [2].

Volvo Car Corporation (henceforth referred to as Volvo) aims to employ fully autonomous driving (AD) cars in traffic. The safety of these cars heavily relies on the performance of multiple sensors in order to make correct decisions. Sensors including radars, cameras, lidar, ultrasonic sensors etc. work together by sensor fusion to interpret the environment around the car. For the output of all these sensors to be trustworthy the sensors must be evaluated in regards to their limitations and statistical sensor models. Evaluating the sensors and their models is important both for verification of autonomous driving and implementation of optimal sensor fusion.

To get insight in the performance of a sensor, another sensor with better accuracy can be used as a reference. With the help of the difference between the sensor of interest and the reference, an error can be estimated. To compare and evaluate the sensors, this project will focus on the high-level outputs from the sensors. That is, outputs after the raw sensor signals have been processed into objects surrounding the ego-vehicle with features describing the object such as velocity, position, convex hull and lifetime represented as multi-dimensional time series. The object matching method and results from [3] will be used as a baseline for what object matching algorithms without machine learning is capable of as it was also conducted at Volvo with the same hardware.

1.2 Aim

The thesis aims to evaluate the possibilities and limitations with machine learning techniques in the application of offline object matching.

The algorithm presented in *Offline object matching and evaluation process for verification of autonomous driving* [3], required time consuming tuning of parameters to work well. The trained models presented in this work are intended to perform at least on par with such an algorithm without requiring any hand tuned parameters.

This thesis also aims to evaluate the performance of different machine learning algorithms on this automotive classification problem. Further the trained models ability to scale between different sensors and noise will be investigated with the aim to find a robust and accurate classifier.

1.3 Scope and boundaries

The models developed will be focused on Volvo's system setup so implementation and signal availability is limited by these systems. Evaluation of performance will be done on datasets provided by Volvo where some pre-processing has already been done. This thesis focuses on implementing an offline matching algorithm and real time matching is outside the scope of this project. As the data is provided by Volvo, the choices of traffic situations, setup of sensors and how the annotated data is labeled is outside the scope of this project. For the offline algorithm to be considered as a feasible solution, it has to be able to train a model and validate it on a personal computer within 12 hours.

1.4 Outline

The remainder of this thesis is organized as follows. In Chapter 2 some preliminaries of the system are covered. Chapter 3 covers in depth the theory and math behind the different parts included in the thesis. In Chapter 4 we go over how the algorithm is set up and how we chose to validate the numerical results that is presented in Chapter 5. Chapter 6 includes comments on various parts in the project and makes some suggestions for future work. Finally, in Chapter 7 a conclusion to the research questions is answered.

2
Preliminaries

In this chapter we introduce the sensor configuration, the datasets and how machine learning combine the two into classifiers.

2.1 Generating high-level objects from sensor measurements

Multiple sensors are involved in a car's interpretation of the world around it. This section aims to introduce the sensors available in the cars used in this project and briefly explain the transformation of raw measurement data into high-level objects.

The first sensor to discuss is the camera. The camera is a passive sensor that captures high resolution images in its field of view. The meaning of passive in this context is that the sensor does not emit any energy, it observes the environment. As a consequence to this, light must be emitted from another source in order for the camera to function. Significant research has been conducted on extracting information about the objects in an image such as [4]. The information extracted through image analysis algorithms is merged with historical information to create time series of features which is part of the inputs to the high-level objects under investigation in this thesis. Though cameras achieve high accuracy in object detection, radial resolution and classification they have limited abilities to detect depth, especially when they work in a monocular setup. In a stereo setup, cameras can achieve better longitudinal accuracy but still inherit the other limitations of cameras such as sensitivity to surrounding light conditions.

The second sensor used in this project is the radar sensor which emits and receives millimeter-waves enabling it to accurately measure longitudinal distance even during harsh weather conditions [5]. For automotive radars, frequencies around 24 GHz and 77 GHz are commonly used as described by J. Hasch et al. [6]. The raw measurement cluster from the radar is merged into objects called tracklets which contain trajectory information such as heading, speed and position.

An important finding here is that the longitudinal accuracy contained in the radar tracklets can be fused together with camera information to gain accuracy in radial and longitudinal measurements. Through the fusion of these measurements, objects with high-level features are created. Note that several radar tracklets can describe one single entity, while the camera only describes it as one.

A third sensor is a Light detection and ranging, (henceforth refereed to as lidar) which is an additional sensor technology that works similar to the radar but emits light, typically with a 905 nm wavelength [7], at fixed angular steps providing excellent accuracy in both radial and longitudinal measurements. In contrast to the radar, the lidar used in this project have the benefit of a 360° horizontal field of view.

2.2 Origin of datasets

This project builds upon prior work done at Volvo and data gathering, annotating and some signal processing was done before the initialization of this project. This section aims to describe the datasets used in this project.

Prior to the initialization of this project only the matching object pairs were annotated but for the methodology suggested in this thesis datasets containing both matching and nonmatching pairs are required. The extraction of nonmatching pairs is covered in the methodology chapter of this report.

There are three different datasets available for matching. The datasets are called RaCam2Lidar, Cam2Lidar and Radar2Lidar. The two latter sets are constructed from RaCam2Lidar as they were created by taking the data from each sensor before the data was fused. The annotations were then inherited through object identification numbers. In all three datasets the reference is the lidar measurements but the sensor under investigation change. The RaCam2Lidar dataset contains sensor-objects with fused information from two radar systems and one camera, the Cam2Lidar dataset contains objects with information from one camera and the Radar2Lidar dataset contains objects with information from two radar systems.

All three datasets were gathered on the same expedition, a route that spanned over Europe and went through Germany, Austria, Switzerland, Italy, France and the UK. This route included complex motorways, country roads, ring roads around large cities and city traffic. The trip can be seen in figure 2.1.

2.3 System setup

The RaCam is mounted in the front windshield while the lidar is mounted on top of the roof. The coordinate system for all latitudinal and longitudinal signals is the same. It has the origin in the center of the rear axle with positive longitudinal axis in the driving direction of the car. Positive latitudinal direction increases in the right to left direction perpendicular to the car's driving direction, see figure 2.2. As the lidar and RaCam are located along the logtitudinal axis, two different offsets to the origin emerge. A compensation for these offsets has been incorporated into the sensor models.

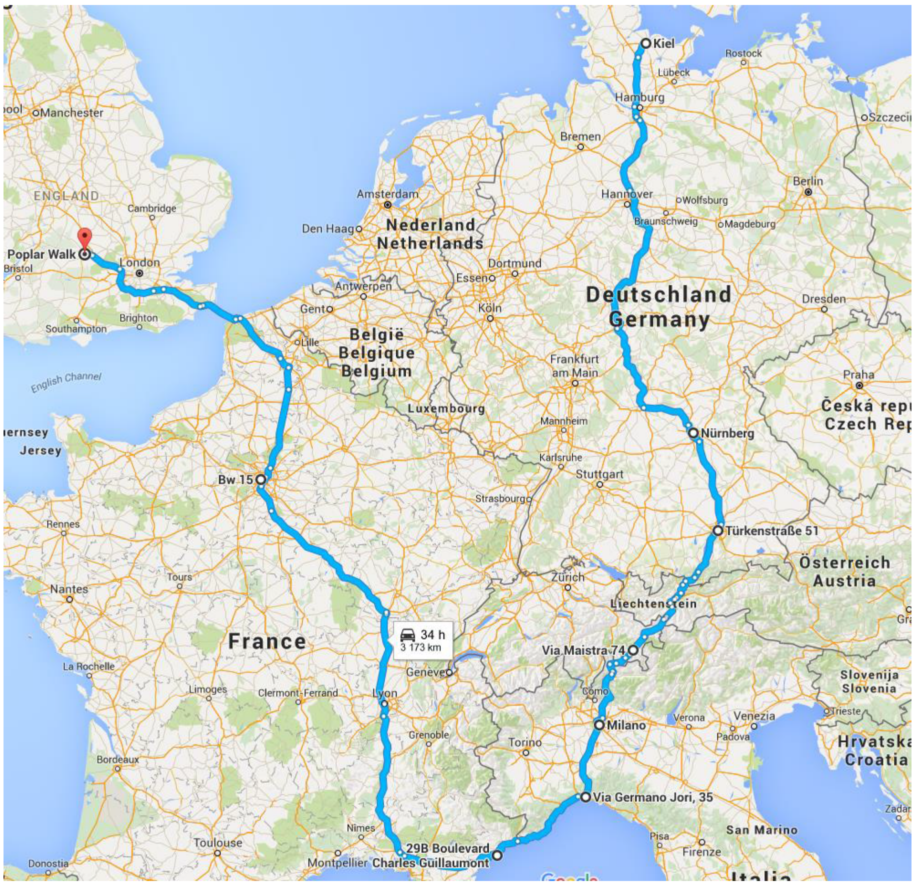


Figure 2.1: Map of the trip taken when gathering data for the datasets used.

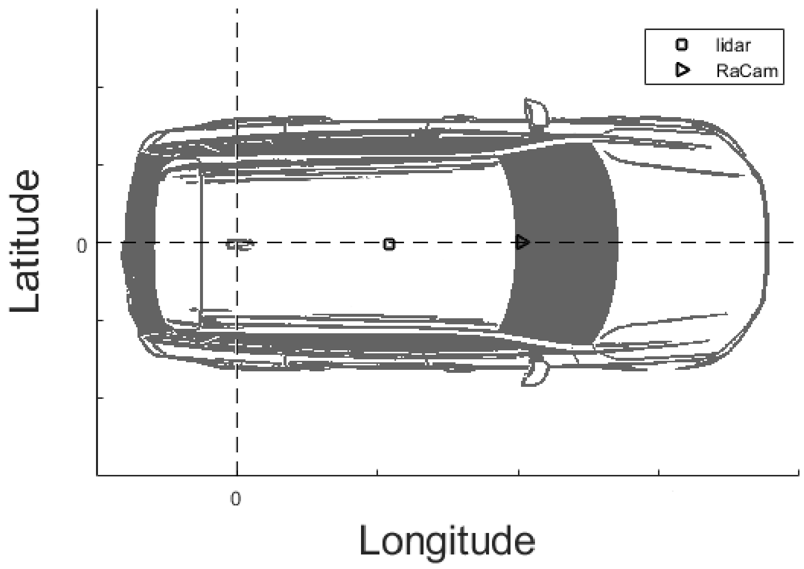


Figure 2.2: Coordinate system definition with origin in the center of the rear axle. The figure also shows that the lidar is placed on the roof, and the RaCam in the windshield of the car.

2.4 Classifying data using machine learning

The objective of this project is to classify objects from different sensor setups as matching or nonmatching pairs. This section introduces the machine learning technologies investigated and applied to classify the objects. This section will give a brief overview, but more in depth explanations are available in the theory chapter, 3.

One of the technologies used within machine learning is Artificial Neural Networks (ANN) which are models inspired by the human brain where a set of neurons are linked with weighted connections and these weights are being adjusted according to some optimization algorithms such as gradient decent [8]. There exists a number of different types of neural networks for example convolutional neural networks which have shown great performance in image classification tasks [9] and recurrent neural networks who excels in applications such as language translation [10]. The main drawback of these algorithms is the difficulty to train them as they require a lot of data to perform well.

The second algorithm introduced was the Support Vector Machine (SVM) which finds a hyperplane that best separates data based on a set of features. This is especially useful in a binary classification setting (where there are only two classes). The benefits of a SVM is that it generally does not overfit, but the drawback is that the input parameters might have to be chosen with more care since it does not handle abstract concepts as well as an ANN [11].

Another machine learning technique of interest in this project is Decision Trees, a technique that traverses down a tree selecting branches based on if statements starting from the root and commits to a classification once it reaches a leaf node [12]. What makes decision trees interesting is the ability to backtrack the results. If a miss-classification happens one can backtrack through the tree to find the source of the error which in this application could give indications of which sensor is providing inadequate information.

The last algorithm to be introduced here is the k-Nearest Neighbor (kNN) [13] which classifies an entry based on a vote taken by the k Nearest Neighbors where k denotes the number of participants in the vote. This is a fairly simple algorithm but has a few cons in that it needs to keep all old data-points to act as neighbors for newer entries resulting in a memory intensive model. In general one of the major drawbacks of the kNN algorithm is that it can be fooled by irrelevant features, but in this case where the features will be carefully selected, this should not be a problem.

3 Theory

In the following sections, the theory behind the methods used to classify objects from the sensor and reference systems as matches or non-matches is introduced. This chapter aims to give a deeper understanding of the methods used and will aid discussion of the results.

3.1 Principal Component Analysis

Principal component analysis (PCA) has been proven useful in feature extraction in conjunction with machine learning applications. Some applications include digit recognition and noise reduction. In this section PCA will be explained by an example followed by the details of how PCA is calculated.

3.1.1 Example and intuition

PCA finds a transformation from the current feature space onto a principal component space where each dimension is orthogonal and spans in the directions of the datasets largest variance. For PCA to be useful in this project an assumption has to hold, which is that features with a high variance have high variance because they capture some dynamics that has a correlation to whether the objects are matching or not. One of the properties of the principal components is that they are ordered in regards to largest variance which is helpful in dimension reduction since the least informative principal components can be disregarded if the corresponding eigenvalue is small.

Let us consider a simple linear example where the sequence $x = \{1, 2, \dots, 100\}$ is transformed through a function

$$f(x) = 2x + 5 \tag{3.1}$$

and the output is then observed by 20 different sensors with a large uncertainty of normally distributed noise with 0 mean and standard deviation of $\sigma = 150$. The feature space has then 20 dimensions where each dimension is the measurements of one sensor. Let's say the task is to classify which measurements from the sequence come from inputs less than 50.

By performing a PCA on the measurements, the data that is captured in a 20 dimensional space is transformed to a new space where the first direction has the largest variance, this direction become the first principal component. The second component becomes the vector that is orthogonal to the first and spans in the

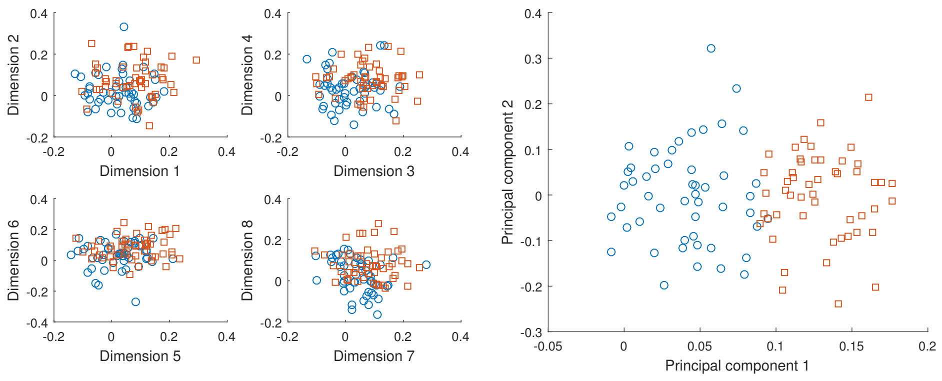


Figure 3.1: Illustration of dimensionality reduction using linear PCA. The blue circles originates from $x < 50$ and the orange from $x > 50$.

direction that has the largest variance and so on. The original data in 20 dimensions can then be transformed by the transformation matrix obtained in the PCA process and projected on a lower dimensional plane.

In figure 3.1 one can see the original 1-8 dimensions shown as 2D scatter plots and the first and second principal components can be seen as a 2D scatter plot to the right. The PCA algorithm has in this case been able to effectively find the underlying dimensionality of the system even though the data was disturbed by large noise. The PCA performed here also shows how dimensionality reduction from 20 to 2 can be done. By only looking at the first principal component, one could easily classify where the circles originated from with a simple threshold value.

3.1.2 Linear PCA

In the example from section 3.1.1, the data was in the form of an 100 by 20 matrix, where the rows represent samples and the columns the 20 different sensors. Lets call this matrix X with dimension m by n. The mapping from the original data into the principal components is mathematically defined as a linear transformation. First, let us center the data matrix X by its column. This is for mathematical convenience when writing the variance in equation 3.4 as we can avoid subtracting the mean μ all the time.

$$\widetilde{X} = (I_m - \tfrac{1}{m}1_m)X \tag{3.2}$$

where I_m is the identity matrix and 1_m is a matrix full of ones (1), both with the size m by m.

Using a set of vectors known as loads $w_k = (w_1, \dots, w_n)^T_{(k)}$ that map each of the rows in matrix $\widetilde{X}$ to principal component column scores $t_{(p)} = (t_1, \dots, t_m)_{(p)}$

$$t_i = x_k \cdot w_k \qquad \text{for} \quad i = 1, \dots, p \quad k = 1, \dots, m \tag{3.3}$$

where $x_{(i)}$ is a row vector from $\widetilde{X}$ then one have to find the loading vectors, w that transform $\widetilde{X}$ that end up in the direction of the maximum variance. Thus the first loading vector has to satisfy

$$\begin{aligned} w_{(1)} = \quad & \max_{w} \quad Var(t_i) = \tfrac{1}{m}\textstyle\sum_i (t_1)^2_{(i)} \\ & \text{subject to} \quad ||w|| = 1 \end{aligned} \tag{3.4}$$

By substituting 3.3 into 3.4 and change to matrix form, we get

$$\begin{aligned} w_{(1)} = \quad & \max_{w} \quad \left\{\tfrac{1}{m}||\widetilde{X}w||^2\right\} = \max_{w}\left\{w^T C w\right\}, \qquad \text{where } C = \tfrac{1}{m}\widetilde{X}^T\widetilde{X} \\ & \text{subject to} \quad ||w|| = 1 \end{aligned} \tag{3.5}$$

At this point, equation 3.5 can be written as in 3.6 where it is recognized as the Rayleigh quotient [14] which is a known engineering problem and has the solution of w being the eigenvectors of C.

$$w_{(1)} = \max_{w}\left\{\frac{w^T C w}{w^T w}\right\} \tag{3.6}$$

The result can easily be confirmed by Lagrange multipliers.

$$\mathcal{L}(w) = w^T C w - \lambda(w^T w - 1) \tag{3.7}$$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = 2Cw - 2\lambda w \tag{3.8}$$

$$\frac{\partial \mathcal{L}(w)}{\partial w} = 0 \quad \Rightarrow Cw = \lambda w \tag{3.9}$$

As C is a square matrix, equation 3.9 can be solved by eigendecomposition. Hense the solution to the problem is eigenvectors w (from 3.9) of C with the largest corresponding eigenvalue will project the data onto the new component space. The first loading vector will be the eigenvector with the largest eigenvalue λ, the second loading vector will then be the eigenvector corresponding to the second largest eigenvalue etc.

The number of loading vectors that is extracted determines the dimensionality reduction as not all the principal components need to be kept. In the example in section 3.1.1, where the original data was in 20 dimensions, by only keeping the first two principal components, a new dimension of 2 was achieved. A point to notice is that C is the covariance matrix of $\widetilde{X}$. There are more ways of calculating the principle components that does not require the use of eigen demoposition but with the help of singular-value decomposition [15].

3.1.3 Kernel PCA (kPCA)

If the data is structured in a more complex and nonlinear way, the standard version of PCA will not be able to represent the nonlinear relation within its linear subspace.

A solution to this is to use a nonlinear transformation $\phi(x)$ from the original space into a new feature space $\mathcal{F}$ where ordinary PCA can be applied. However, this can be very computational heavy and inefficient. Luckily there is a way to use kernel methods (described more in section 3.3) to avoid having to calculate $\phi(x)$ explicitly [16].

First, consider the nonlinear map

$$\begin{aligned}\Phi:\ & \mathbb{R}^N \to \mathcal{F}, \quad \text{where} \quad \widetilde{X} \in \mathbb{R}^N \\ & \widetilde{X} \to \mathcal{X}\end{aligned} \tag{3.10}$$

and assume that $\mathcal{X}$ is centered. Then the covariance matrix in $\mathcal{F}$ looks like

$$\mathcal{C} = \frac{1}{m}\sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T, \quad \text{where } x_i \text{ is a vector in } \widetilde{X}. \tag{3.11}$$

$$\widetilde{X} = \begin{bmatrix} \dots & x_1^T & \dots \\ \dots & \vdots & \dots \\ \dots & x_i^T & \dots \end{bmatrix}$$

And the eigenvectors

$$\mathcal{C}w_k = \lambda w_k. \tag{3.12}$$

By combining equation 3.11 and 3.12, we get

$$\frac{1}{m}\sum_{i=1}^{m} \phi(x_i)\phi(x_i)^T w_k = \lambda w_k, \tag{3.13}$$

where the eigenvectors can be expressed as linear combination of features

$$w_k = \sum_{i=1}^{m} \alpha_i \phi(x_i). \tag{3.14}$$

Now by substituting 3.14 into 3.13 we get

$$\frac{1}{m}\sum_{i=1}^{m} \phi(x_i) \sum_{j=1}^{m} \alpha_j \phi(x_i)^T \phi(x_j) = \lambda \sum_{i=1}^{m} \alpha_i \phi(x_i) \tag{3.15}$$

Note that the following is true, (see appendix A for proof)

$$\phi(x)\phi(x)^T w = \{\phi(x) \cdot w\}\phi(x)^T. \tag{3.16}$$

We define a kernel function as

$$\kappa(x_i, x_j) := \phi(x_i)^T \phi(x_j), \tag{3.17}$$

and by multiplying 3.15 from the left with $\phi(x_l)^T$, for all $l = 1, \dots, m$. We get an expression that can be written using the kernel function from equation 3.17 and the product from 3.16.

$$\frac{1}{m}\sum_{i=1}^{m} \phi(x_l)^T \phi(x_i) \sum_{j=1}^{m} \alpha_j \phi(x_i)^T \phi(x_j) = \lambda \sum_{i=1}^{m} \alpha_i \phi(x_l)^T \phi(x_i)$$

$$\frac{1}{m}\sum_{i=1}^{m} \kappa(x_l, x_i) \sum_{j=1}^{m} \alpha_j \kappa(x_i, x_j) = \lambda \sum_{i=1}^{m} \alpha_i \kappa(x_l, x_i) \tag{3.18}$$

Rewriting the sums over the kernel functions as the m by m matrix $\mathbf{K} = \kappa(x_i, x_j)$ and $\mathbf{a}$ as a column vector with all the α entries, $\mathbf{a} = [\alpha_1, \dots, \alpha_m]^T$, we get equation

3.19 which can be solved by having one $\mathbf{K}$ removed from each side. Note that all solutions of 3.20 also satisfies 3.19 [17].

$$\frac{1}{m}\mathbf{K}^2\mathbf{a} = \lambda\mathbf{a}\mathbf{K}. \tag{3.19}$$

$$\mathbf{K}\mathbf{a} = m\lambda\mathbf{a} \tag{3.20}$$

From the beginning we have assumed that the projected data in $\mathcal{F}$ is centred. Even though the input data $\widetilde{X}$ is centered, the mapping done by Φ to the feature space leaves no guarantee that the mapped data will be centred. As we want to avoid working in the feature space and utilize the kernel method instead, the projected centralized data points $\widetilde{\phi}(x_i)$ has to be expressed in terms of the kernel function 3.17. Substitute centralizing from equation 3.2 into 3.21.

$$\begin{aligned}
\widetilde{\mathbf{K}}_{i,j} =& \widetilde{\phi}(x_i)^T\widetilde{\phi}(x_j) && (3.21)\\
=&(I_m - \frac{1}{m}1_m)\phi(x_i)^T \quad (I_m - \frac{1}{m}1_m)\phi(x_i)\\
=&\Big(\phi(x_i)^T - \sum_{l=1}^{m}\phi(x_l)^T\Big)\Big(\phi(x_j) - \sum_{j=1}^{m}\phi(x_j)\Big)\\
=&\phi(x_i)^T\phi(x_j) - \frac{1}{m}\sum_{l=1}^{m}\phi(x_i)^T\phi(x_l) - \frac{1}{m}\sum_{l=1}^{m}\phi(x_l)^T\phi(x_j)\\
&+\frac{1}{m^2}\sum_{j=1}^{m}\sum_{l=1}^{m}\phi(x_j)^T\phi(x_l)\\
=&\kappa(x_i,x_j) - \frac{1}{m}\sum_{l=1}^{m}\kappa(x_i,x_l) - \frac{1}{m}\sum_{l=1}^{m}\kappa(x_l,x_j) + \frac{1}{m^2}\sum_{j=1}^{m}\sum_{l=1}^{m}\kappa(x_j,x_l) && (3.22)
\end{aligned}$$

Expressed in matrix notation this is equivalent to

$$\widetilde{\mathbf{K}} = \mathbf{K} - \frac{1}{m}\mathbf{1}_m\mathbf{K} - \mathbf{K}\frac{1}{m}\mathbf{1}_m + \frac{1}{m^2}\mathbf{1}_m\mathbf{K}\mathbf{1}_m \tag{3.23}$$

where 1_m is a matrix of dimension m by m with all entries equal to one.

Now the principal components can be calculated by combining the results from above to avoid computing $\phi(x)$ explicitly.

$$\sum_{i=1}^{m}\widetilde{\alpha}_i\widetilde{\kappa}(x,x_i) \tag{3.24}$$

To summarize, first create the matrix $\mathbf{K}$ (3.17) and use it to create the centered kernel matrix $\widetilde{\mathbf{K}}$ (3.23). Then find the eigenvectors to $\widetilde{\mathbf{K}}$ from (3.20). With the eigenvectors and eigenvalues, extract kernel principal components with the use of equation 3.24.

3.2 Support Vector Machine

This section aims to explain the main ideas behind support vector machines (SVM). In its simplest form the SVM finds the hyperplane that best separates linearly separable data of two classes [11]. The datapoints x_i are of some p dimension such that $x_i \in \mathbb{R}^p$, $i \in \{1, 2 \dots N\}$ where N is the number of datapoints to separate. The class is denoted $y \in \{-1, 1\}$ and the dataset D can then be defined as

$$D = \{(x_i, y_i) | x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^{N} \tag{3.25}$$

The SVM will try to seperate the data with a hyperplane as illustrated in the 2D example in figure 3.2. In the 2D example the hyperplanes are lines

$$y = ax + b \Rightarrow \begin{bmatrix} a \\ -1 \end{bmatrix}^T \begin{bmatrix} x \\ y \end{bmatrix} + b = 0 \Rightarrow \mathbf{w}^T x + b = 0 \tag{3.26}$$

that can be generalized into the hyperplane formula $\mathbf{w}^T x + b = 0$ which will be used as the formula for the separating hyperplane throughout this section. As can be seen in figure 3.2 there may be different hyperplanes that separate the classes and the optimization task for the SVM is to find the hyperplane that maximizes the margin. By eyeballing the right figure of 3.2, a naive approach could be to introduce a hyperplane in the middle of the two red lines. Eyeballing the problem works in a 2D setup but when dimensionality increases it is no longer possible and the solution found is rarely optimal. The margin refers to the shortest Euclidean distance from any datapoint x_i to the hyperplane. The scaling of the data does not effect the linear separability so from now on $\mathbf{w}$ will refer to a vector perpendicular to the hyperplane of unit length.

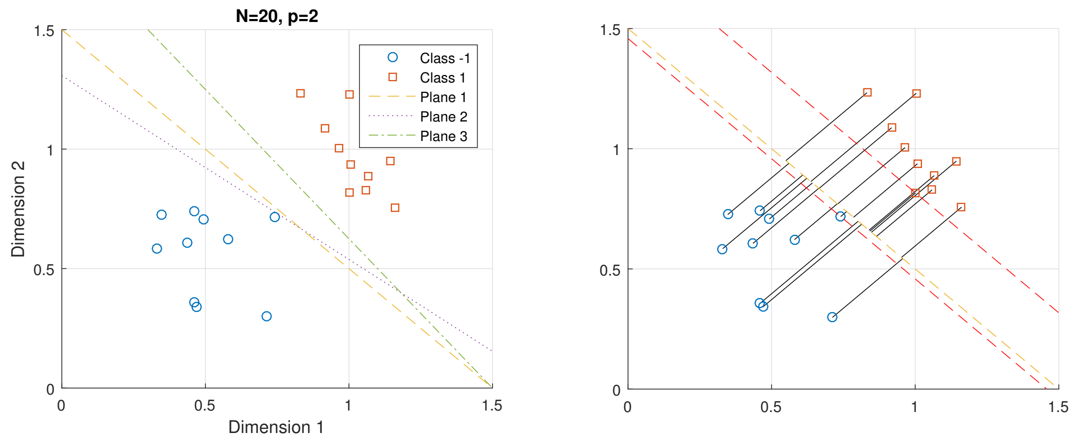


Figure 3.2: Data of class -1 and 1, to the left separated by some hyperplane. In the right figure the margin from plane 1 to the datapoints has been marked by a black vector and red lines have been drawn to indicate the total margin.

Since the SVM is supposed to classify the samples, a formula for classification is needed. Everything on the right side of the plane should be positive and everything on the left side should be negative.

$$\begin{cases} \mathbf{w}^T x_i + b > 0 & \text{if } y_i = 1 \\ \mathbf{w}^T x_i + b < 0 & \text{if } y_i = -1 \end{cases} \tag{3.27}$$

3.2.1 Finding the first constraint

The constraint in equation 3.27 could classify samples of either side of a plane but it does not help in finding the optimal separating hyperplane since no separating region has been introduced. By requiring

$$\begin{cases} \mathbf{w}^T x_i + b \geq 1 & \text{if } y_i = 1 \\ \mathbf{w}^T x_i + b \leq -1 & \text{if } y_i = -1 \end{cases} \tag{3.28}$$

a margin where no samples are located is introduced. This margin is the distance between the two red lines in figure 3.2. The requirement can be rewritten as

$$y_i(\mathbf{w} \cdot x_i + b) - 1 \geq 0 \tag{3.29}$$

which will be used as a constrain in the optimization problem.

3.2.2 Maximize the margin as an optimization problem

A point on the lower red line in figure 3.2 would satisfy

$$\mathbf{w} \cdot x_i + b + 1 = 0 \tag{3.30}$$

and by taking a step of length m in the perpendicular direction to this plane towards the upper red line a point on that line would satisfy

$$\mathbf{w} \cdot (x_i + m\mathbf{w}) + b - 1 = 0 \tag{3.31}$$

where m becomes the margin which we seek to maximize. Putting equation 3.30 and 3.31 together gives

$$\begin{aligned} \mathbf{w} \cdot x_i + b + 1 &= \mathbf{w} \cdot (x_i + m\mathbf{w}) + b - 1 \\ m &= \tfrac{2}{||\mathbf{w}||^2} \end{aligned} \tag{3.32}$$

where optimizing for maximizing the margin has the same solution as minimizing the reciprocal which will be used when setting up this constraint optimization problem [18].

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \tfrac{1}{2}||\mathbf{w}||^2 \\ \text{subject to} \quad & y_i(\mathbf{w} \cdot x_i + b) - 1 \geq 0 \end{aligned} \tag{3.33}$$

By changing to a Lagranian formulation of the problem, the constraint is replaced with a series of Lagrange multipliers α_i. The Lagrange multipliers are subjected to constraint which is easier than the above constraint to handle but more importantly results in the property that the training data will only appear in the form of dot products between vectors which is crucial for generalizing to the nonlinear case using kernel methods described in section 3.3. With the Lagranian formulation

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}||\mathbf{w}||^2 - \sum_{i=1}^{N} \alpha_i [y_i(\mathbf{w} \cdot x_i + b) - 1] \tag{3.34}$$

the duality principle can be utilized for this convex optimization problem where both Slater's condition and strong duality holds. By utilizing the duality principle the Wolfe dual Lagrangian function can be found which results in the Wolfe dual problem

$$\begin{aligned} \max_{\alpha} \quad & \textstyle\sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j x_i \cdot x_j \\ \text{subject to} \quad & \alpha_i \geq 0 \quad \forall i \in 1 \dots N \\ & \textstyle\sum_{i=1}^{N} \alpha_i y_i = 0 \end{aligned} \tag{3.35}$$

which can be solved by a QP (Quadratic Programming) solver [19] and thus finding the optimal separating hyperplane.

3.2.3 Dealing with noise and nonlinearity

In many real world scenarios the data might be affected by noise which makes a linear separation impossible. To handle these scenarios slack variables ξ_i will be introduced which are designed to penalize samples within the margin through some function f [20]. The amount of penalty is configured using a design parameter C. The optimization problem with these slack variables included becomes

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 + C \textstyle\sum_{i=1}^{N} f(\xi_i) & \\ \text{subject to} \quad & y_i(\mathbf{w} \cdot x_i + b) - 1 + \xi_i \geq 0 & \forall i \\ & \xi_i \geq 0 & \forall i \end{aligned} \tag{3.36}$$

Introducing the slack variables improves handling noisy data but when the data is not linearly separable as in figure 3.3 another solution is needed namely increasing the dimensionality. The dimensionality is increased using the kernel trick as described in section 3.3 which might lead to linearly separable data in higher dimensions. Figure 3.3 shows a 2D dataset to the left which can not be linearly separated but when transformed into a 3D dataset using a similarity function based on the radial basis kernel

$$\kappa(x, x') = e^{-\dfrac{\|x - x'\|^2}{2\sigma^2}} \tag{3.37}$$

the separation becomes an easy task [21]. In figure 3.3 the σ parameter has been set to 0.6 and a plane at $z = 0.7$ has been added as an example of a separating plane. The σ parameter will be called kernel scale throughout this report. In literature and in this report both the names Gaussian kernel and radial basis kernel are used to describe the same kernel function.

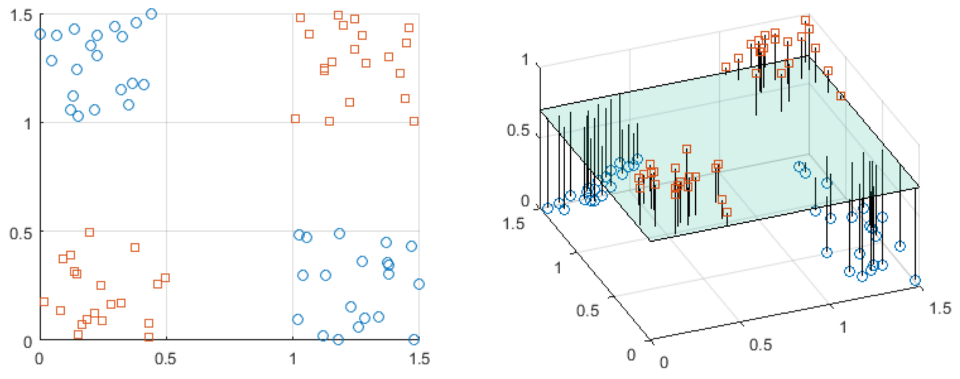


Figure 3.3: Dataset of linearly inseparable data to the left. The same dataset is transformed by the kernel trick to a higher dimension where they are linearly separable to the right.

3.3 Kernel trick

This section aims to give a brief introduction to the kernel trick used in both SVM and kPCA. In short it is an approach where one uses a kernel function in order to substitute for a inner product between data to end up in a high dimension.

If we have data that we are not linearly able to classify but we can describe it as inner products, then we can use a mapping ϕ, from the input space into a higher dimension feature space $\mathcal{F}$ where the data is linearly separable, see figure 3.3.

$$\begin{aligned} \mathbf{a} = &\quad [a_1, a_2]^T \rightarrow \phi(a) = [a_1^2, a_2^2, a_1a_2, a_2a_1]^T \\ \mathbf{b} = &\quad [b_1, b_2]^T \rightarrow \phi(b) = [b_1^2, b_2^2, b_1b_2, b_2b_1]^T \end{aligned} \tag{3.38}$$

Lets consider an example where the input space of 2-D is mapped by $\phi(\cdot)$ to a feature space of 4-D as in equation 3.38. After the mapping, the inner product becomes:

$$\langle \phi(\mathbf{a}), \phi(\mathbf{b}) \rangle = \phi(\mathbf{a})^T \phi(\mathbf{b}) = a_1^2 b_1^2 + a_2^2 b_2^2 + 2a_1a_2b_1b_2 \tag{3.39}$$

This process can also be recognized as taking the squared inner product between **a** and **b**.

$$\langle \mathbf{a}, \mathbf{b} \rangle = (\mathbf{a}^T \mathbf{b})^2 \tag{3.40}$$

We have now shown that the square of this inner product is the inner product in the feature space thus verifying the kernel $\kappa(\mathbf{a}, \mathbf{b}) = (\mathbf{a}^T \mathbf{b})^2$. This is known as a polynomial kernel of second degree [22], see table 3.1 for more popular kernels.

Kernel name	$\kappa(\mathbf{x}, \mathbf{y})$	**Comment**
Polynomial	$(a^T b + c)^d$	Constant $c > 0$
Gaussian Radial Basis Function (RBF)	$e^{-\frac{\|\|\mathbf{x}-\mathbf{y}\|\|^2}{2\sigma^2}}$	Design parameter $\sigma > 0$
Laplacian	$e^{-\alpha\|\|\mathbf{x}-\mathbf{y}\|\|}$	Design parameter $\alpha > 0$
Abel	$e^{-\alpha\|\mathbf{x}-\mathbf{y}\|}$	Design parameter $\alpha > 0$
Sigmoid	$tanh(\gamma \cdot \mathbf{x}^T \mathbf{y} + r)$	Design parameter γ and r

Table 3.1: Table of common kernel functions.

With that said, there exists many more valid kernels and one does not have to find the map ϕ to verify it. The requirements for a kernel to be valid can be stated through two theorems [23]:

Theorem 1 Let $\kappa(x, y)$ be a real symmetric function on a finite input space, then it is a kernel function if and only if the matrix K with components $\kappa(x_i, x_j)$ is positive semi-definite.

Theorem 2 (Mercer's theorem) If $\kappa(x, y)$ is a continuous symmetric kernel of a positive integral operator T, i.e.

$$(Tf)(\mathbf{y}) = \int_C \kappa(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) d\mathbf{x} \tag{3.41}$$

with

$$\int_{C \times C} \kappa(\mathbf{x}, \mathbf{y}) f(\mathbf{x}) f\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \tag{3.42}$$

for all $f \in L_2(C)$ then it can be expanded in a uniformly convergent series in the eigenfunctions ψ_j and positive eigenvalues λ_j of T , thus:

$$\kappa(\mathbf{x}, \mathbf{y}) = \sum_{j=1}^{n_e} \lambda_j \psi_j(\mathbf{x}) \psi_j(\mathbf{y}) \tag{3.43}$$

where n_e is the number of positive eigenvalues.

3.4 Artificial Neural Networks

Artificial Neural Networks (ANNs) are machine learning models inspired by the human brain which consists of a network of neurons. This section aims to explain the neural network model and mathematical strategies involved in making these models computationally efficient and introduce the design choices involved.

A neuron will be modeled as a sum of weighted inputs sent through an activation function.

$$\sigma(\sum_k w_k a_k + b) \tag{3.44}$$

In equation 3.44, a_i denotes inputs to the neuron, w_i denotes the weights, b denotes bias and σ denotes the activation function.

Lets consider the simplest possible network consisting of only 2 neurons, with one input to each.

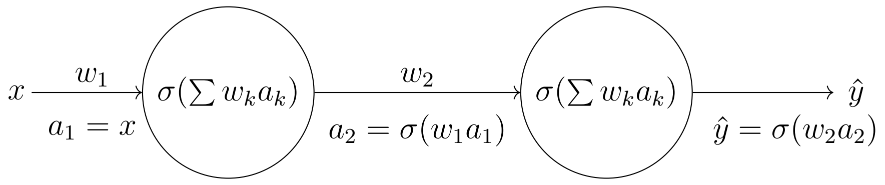


Figure 3.4: Showing a small neural network where w_i denotes the weight and a_i denotes the signal at the edge. x denotes the input to the network and $\hat{y}$ the output.

In figure 3.4 a network that takes an input x and feeds it forward to produce an output $\hat{y}$ depending on the weights is shown. The weights will be changed so that the network approximates some function which is generally unknown. The idea is to give the network a set of training samples for which the output is known and adjust the weights until the output of the network is close to the sampled output of the function one seeks to approximate. Once the model has been trained it can be used to estimate the output of new input data.

3.4.1 Gradient descent

To change the weights some kind of cost function is needed to evaluate the performance of the network with the current weights. The cost function should be differentiable so that the partial derivatives with respect to the weights can be used for gradient descent. For conciseness lets introduce the notation $h(x_i, \mathbf{w}) = \hat{y}_i$ for the neural network where $\hat{y}_i$ is the predicted output for some input x_i given the weights $\mathbf{w}$. One choice of cost function is the quadratic loss function

$$L(w) = \sum_i (h(x_i, \mathbf{w}) - y_i)^2 \tag{3.45}$$

where y_i is the true output of the function the network seeks to approximate. The gradient of the cost function becomes

$$\frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}) = \sum_i 2(h(x_i, \mathbf{w}) - y_i) \frac{\partial}{\partial \mathbf{w}} h(x_i, \mathbf{w}). \tag{3.46}$$

With a simple activation function $\sigma(w, a) = w \cdot a$ the partial derivative of the loss function can be solved by first calculating

$$\begin{aligned} \tfrac{\partial}{\partial \mathbf{w}} h(x_i, \mathbf{w}) &= \left(\tfrac{\partial}{\partial w_1} h(x_i, w_1), \tfrac{\partial}{\partial w_2} h(x_i, w_2)\right) = \\ \left(\tfrac{\partial}{\partial w_1} w_2 w_1 x_i, \tfrac{\partial}{\partial w_2} w_2 w_1 x_i\right) &= \left(w_2 x_i, w_1 x_i\right) \end{aligned} \tag{3.47}$$

and then by using the results from equation 3.47 in 3.46 the gradient can be found as

$$\Delta_{\mathbf{w}} L = \Big(\sum_i 2 w_2 x_i (\hat{y}_i - y_i), \sum_i 2 w_1 x_i (\hat{y}_i - y_i)\Big) \tag{3.48}$$

which can be used to update the weights according to

$$\mathbf{w} = \mathbf{w} - \eta\Delta_{\mathbf{w}}L \tag{3.49}$$

where η is the step size which is a design parameter. A small η causes the model to converge very slowly and a large might cause the model to become unstable. In this project η will be set dynamically so that large steps are taken in the beginning to speed up training and as the model settles on a solution the step size will be smaller to gain stability.

3.4.2 Activation functions

In the gradient descent section a very simple activation function was assumed. The activation function is a design choice so a few alternatives will be presented here.

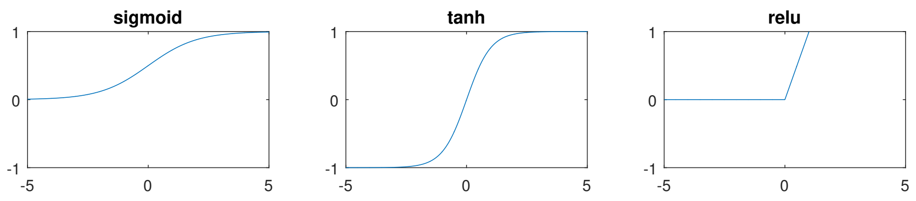


Figure 3.5: Plot of three activation functions sigmoid, tanh and relu.

The first activation function to discuss is the sigmoid function shown to the left in figure 3.5 which takes any real value and maps it between 0 and 1. The formula for the function is

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{3.50}$$

and as explained in section 3.4.1 the derivative of the activation function is part of the gradient descent used to update the weights so it is important that the derivative can be computed in an efficient way. Looking at the derivative of the sigmoid function gives

$$\begin{gathered}\tfrac{d}{dx}\sigma(x) = \tfrac{d}{dx}(1+e^{-x})^{-1} = -(1+e^{-x})^{-2}(-e^{-x}) = \\ \tfrac{e^{-x}}{(1+e^{-x})^2} = \tfrac{1+e^{-x}-1}{(1+e^{-x})^2} = \tfrac{1+e^{-x}}{(1+e^{-x})^2} - \tfrac{1}{(1+e^{-x})^2} = \tfrac{1}{(1+e^{-x})} - \tfrac{1}{(1+e^{-x})}^2 = \\ \sigma(x)(1-\sigma(x))\end{gathered} \tag{3.51}$$

which is an important finding since the derivative can be represented as one simple multiplication operation and one subtraction operation with respect to the function value. In a neural network setting this means that if the value of $\sigma(x)$ is saved when propagating forward through the network the gradients can be calculated efficiently.

The second function to discuss is the tanh function which is the hyperbolic tangent [24] shown in the middle of figure 3.5. The function takes a value and maps it to a value between -1 and 1. One could use the derivations of the sigmoid function for the tanh by letting

$$tanh(x) = 2\sigma(2x) - 1 \tag{3.52}$$

and thus the same computationally efficient properties are preserved.

The third activation function to discuss is ReLU, short for rectified linear units. It is defined as

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \tag{3.53}$$

and can be seen in figure 3.5 to the right. For ReLU the derivative is 1 for $x > 0$ and 0 for $x < 0$ which is also computationally efficient. Using ReLU has shown increased performance and reduced training time [25] [26] but may cause dead neurons since some weight might be changed during training so that a neuron never fires again [27].

3.4.3 Expanding the network

The network presented in figure 3.4 is a highly simplified version of a network with only two weights and all biases set to zero to simplify derivations. Using such a simple network does not provide any useful results in this project so it will be expanded in this section. In the previous subsections two design parameters have been highlighted namely the choice of step size in gradient descent and the choice of activation function. This section aims to introduce the third design parameter which is the design of the network and the fourth which is the batch size. Detailed derivations of the algorithms used will not be part of this section as they are documented in the references.

Neural networks follow a column wise layering of neurons where the neurons in a column are connected to some or all (fully connected) neurons in the neighboring columns. As a naming convention N-Layer networks refers to the number of columns in a network where the input is not counted. The size of a network will also be described in terms of number of weights. In figure 3.6 an example of a 2-Layer network with n inputs and m outputs is shown.

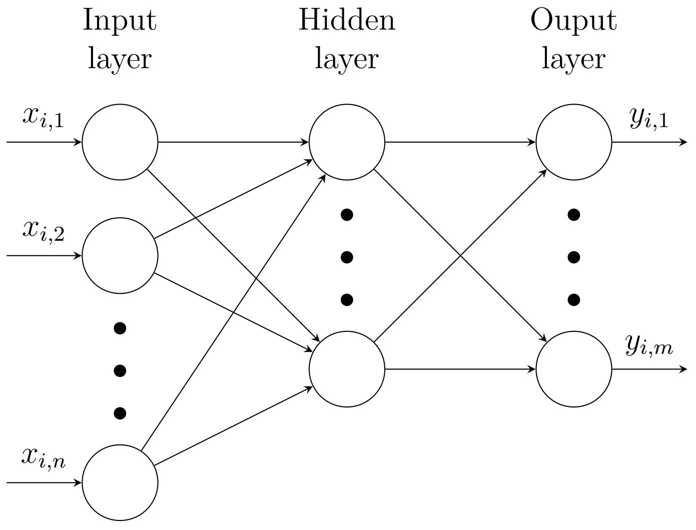


Figure 3.6: The figure shows a 2-Layer fully connected feed forward network

The structure of the network is a design parameter and should be chosen according to the application, for instance in image classification networks of millions of neurons have been used in 10-20 layers [26] [9] to capture the hierarchical structure of objects in images.

To update the weights associated with all those neurons an algorithm called back propagation [28] is used which preforms an approximation of gradient descent described in section 3.4.1 and utilizes the efficiently computed derivatives of the activation functions described in section 3.4.1. A few tricks are introduced in the algorithm to handle large dataset. From section 3.4.1 the loss function was defined as the sum of the squared error of all samples which is not feasible with large datasets so instead another parameter called the batch size is introduced. The batch size is the number of samples to consider in each iteration of the gradient descent in the back propagation algorithm. Recall from equation 3.45 that the loss is the sum over all samples i but in back propagation a subset of the samples is used and the size of the subset is called batch size.

3.4.4 Training difficulties

Solid results have been achieved using deep neural networks [26] but they are hard to train properly and come with a couple of issues that affect the design choices which will be addressed in this section [29].

The unstable gradient problem will be used as a term for both the vanishing gradient problem and exploding gradient problem where the gradient will get small as it is propagated back through the network according to the back propagation algorithm causing very small updates to the weights in the earlier layers or get large causing too large updates in the earlier layers.

It has been shown that the unstable gradient problem can be addressed with proper initialization of the network [30] and by considering the choice of activation function and their respective derivatives as presented in section 3.4.2 [31].

3.5 Decision Trees

This section will describe decision trees as a machine learning strategy for classification. The topics include building trees recursively, constructing a loss function and tuning the size of the tree.

The underlying strategy for decision trees is to ask true or false questions until either the algorithm is confident in its classification or it is not allowed to ask more questions. The amount of questions allowed is a design parameter referred to as the maximum size of the tree. The questions to ask will be decided by traversing down the tree where each node contains a question and the answer to this question will decide which branch to take. The traversing stops once a leaf node is reached and the data point is predicted to be of the class associated with that leaf node.

3.5.1 Training and building a tree

The first node called the root receives the whole dataset denoted D. The dataset is split by a true or false statement. To decide on a statement the algorithm will go through every entry in D and for every entry it will go through every feature and split D by the value of that feature. Each split will be evaluated by the information gain function (see section 3.5.2) and the split with the highest gain will be chosen as the statement of the root node. The split is then performed and the dataset D is divided into two sets D_0 and D_1 where D_1 contain the entries where the split statement evaluates to true and D_0 contains everything else. The datasets are then sent to their respective child node. For each child node every possible split in the divided dataset will be evaluated and the split that produces the most information gain will once again be chosen. This strategy is repeated until some stopping criteria is met (see section 3.5.3). Once some stopping criteria is met the node becomes a leaf and gets associated with a class, in this thesis a match or nonmatch. The class of the node is determined by the most common label of the dataset in the node.

3.5.2 Information gain and Gini impurity

To decide which split to chose when building the tree a function to evaluate the effectiveness of a split is needed. In this section the Gini impurity (equation 3.54) function will be introduced as a loss function and information gain will be defined as the decrease in Gini impurity for a particular split. There exists alternatives to the Gini impurity function but this section will not discuss them since empirical results [32] have not been able to determine if there is a benefit of using them over the Gini impurity.

The Gini impurity function evaluates how likely two randomly selected samples from the dataset at a node are to be of different classes. This is calculated as

$$Gini(n) = 1 - \sum_i p(i|n)^2 \quad i \in \{0, 1\} \tag{3.54}$$

where n is the dataset at the node, i spans over all labels and $p(i|n)$ is the probability of randomly picking a sample of label i from the dataset n. Since this thesis focuses on a binary classification problem i only takes two values where 0 corresponds to nonmatching and 1 corresponds to matching samples.

The information gained by preforming a particular split is defined as

$$\mathrm{IG}(n, l, r) = Gini(n) - \frac{|l|}{|n|} Gini(l) - \frac{|r|}{|n|} Gini(r) \tag{3.55}$$

where n is the dataset at the node, l is the dataset in the left branch after the proposed split and r is the dataset in the right branch. Note that the impurity is weighted by the fraction of the samples since it is better to classify a large number of samples correctly than a few. The split that produces the maximum information gain (IG) is chosen at each node.

3.5.3 Stopping criteria and pruning

At some point the splitting of the nodes have to stop so that the algorithm terminates. If the splitting stops too early performance suffers since the leaves are impure, if the splitting stops too late the model is likely to overfit.

The main stopping condition is if a node is pure, that is when all samples in the node are of the same class. The node will then be marked as a leaf and will not be split. There exists a number of optional stopping conditions that may be used to limit the size of the tree. These are presented in table 3.2.

Name	**Description**
Purity threshold	If a node reaches a predefined purity this can be used as a stopping condition.
Samples threshold	One could implement a limit on the minimum amount of samples in a node. If the number of samples goes below this value the splitting is stopped.
Maximum size	If the allowed maximum number of nodes is reached that can be used as a stopping condition.
Maximum depth	If there is a defined maximum number of steps from the root to a leaf the splitting is stopped when this depth is reached.
Feature equality	If all the features are equal no rule can be constructed to split the data.
Validation testing	By using a validation dataset splitting can be stopped when the performance on the validation set is not improving anymore.

Table 3.2: Table of stopping conditions for decision trees.

Implementing all or some of the stopping criteria presented in table 3.2 can produce working decision trees but occasionally stopping criteria tend to suffer from lack of global knowledge [33] and some of them introduce additional design parameters which then have to be tuned. An alternative to using stopping criteria is to prune the trees instead.

When using pruning the first step is to let the tree fully grow until every leaf node is pure. Pruning then removes subtrees until the desired size of the tree is reached. In this thesis reduced error pruning has been used. Reduced error pruning is an algorithm where the summed error over a subtree is compared to the received error if the subtree was converted to a leaf node with the majority class label. If the conversion from subtree to leaf node is not harmful then the tree is pruned and the subtree is converted into a leaf node with the majority class label. The error is evaluated on a validation dataset and subtrees are compared in a bottom up manner as suggested by Elomaa and Kääriäinen [34].

3.5.4 Boosting

Boosting is a technique used to combine weak classifiers in a meaningful way to increase the overall accuracy [35]. Weak classifiers are classifiers that classify the dataset better than random chance. The boosting algorithm used in this thesis is called Adaboost. Boosting can be done with any weak classifier but more complex classifiers will take longer to train, especially in a boosting setup since many classifiers have to be trained and weighted together in the boosting setup. In this section the weak classifiers are assumed to be a decision trees of size 20.

Let x_i be sample i and y_i be the corresponding label where $i = 1, ..., m$. In this binary classification setup $y_i \in \{-1, +1\}$ where -1 correspond to a nonmatching sample and $+1$ corresponds to a matching sample. Each weak classifier will be denoted $h_t(x)$ and the number of weak classifiers weighted together is denoted T, in this thesis a T value of 30 has been used. Pseudo code for adaboost is found in algorithm 1.

```
input  : Samples $x_i \in X$ and corresponding labels $y_i \in Y$
output: Classifier $H(x)$
Initialize: $D_1(i) = 1/m$ for $i = 1, ..., m$
for $t = 1$ to $T$ do
    $h_t$ = train tree using samples drawn using distribution $D_t$
    $\hat{y} = h_t(X)$
        $\epsilon_t = \mathbf{Pr}_{i \sim D_t}[\hat{y}_i \neq y_i]$    (3.56)
        $\alpha_t = \frac{1}{2}\ln\Big(\frac{1-\epsilon_t}{\epsilon_t}\Big)$    (3.57)
    for $i = 1$ to $m$ do
        $D_{t+1}(i) = \frac{D_t(i)exp(-\alpha_t y_i \hat{y}_i)}{Z_t}$    (3.58)
    end
end
```

$$H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{3.59}$$

Algorithm 1: Pseudo code of adaboost algorithm

In equation 3.56 the error is calculated as the probability of a sample being classified incorrectly when drawing samples from the distribution D_t, in other words the weighted error. In equation 3.58 Z_t is a scaling factor to make $D_{t+1}(i)$ a distribution.

One property of adasboost is that it focuses more on the samples that are hard to solve, equation 3.58 changes the distribution so that failed samples become more important and the next weak learner (h_{t+1}) will be trained more on these hard samples.

A second property of adaboost comes from equation 3.57 where each weak classifier is weighted based on performance, in figure 3.7 one can see that a classifier with

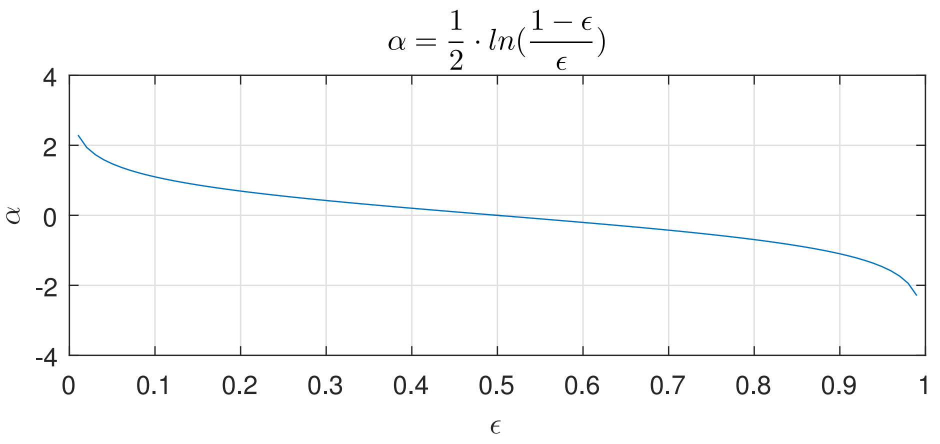


Figure 3.7: Figure of classifier weights associated as a function of weighted error.

weighted error of 0.5 (random guess) will be given a weight of 0 and not be taken into account in the combined classifier. Also note that a classifier that performs worse than random guessing will be given a negative weight essentially making it perform better than random guessing.

The third propert of adaboost to highlight is the distribution built by equation 3.58. Since failed samples are given a higher weight in the distribution these samples will be picked more often when training the next tree. This has the effect that adaboosting focuses more on hard samples which can be beneficial because hard problems require more attention but can be harmful on noisy datasets [36].

3.5.5 Bagging

Bagging or Bootstrap aggregating is a technique used to reduce the variance in the learned representations of the dataset [37]. It works by training T different classifiers, in this case trees, and train them using N datapoints drawn from the original input data X with replacement. Once all T classifiers have been trained they classify new datapoints by taking a vote and the most common class is returned. In algorithm 2 the pseudo code for bagging can be found. In this thesis T will be set to 30 and the number of splits in each tree has be set to 20. N has been set to the number of entries in X.

```
input  : Samples $x_i \in X$, corresponding labels $y_i \in Y$, Number of samples to
         draw N
output: Classifier $H(x)$
for $t = 1$ to $T$ do
    $\bar{X}$ = empty set
    while $|\bar{X}| < N$ do
        $x \leftarrow$ random sample from X
        $\bar{X}$.add($x$)
    end
    $h_t$ = train tree using $\bar{X}$
end
$H(x) = \text{sign}(\sum_{t=1}^{T} h_t(x))$
```

Algorithm 2: Pseudo code of bagging algorithm

3.6 K-nearest-neighbor

In this section the machine learning algorithm K-nearest-neighbor will be introduced. This is a simple algorithm that keeps all the datapoints in the training data and when making a classification a vote will be taken among the K closest neighbors. The datapoint will be classified by the majority class of the neighbors.

The idea is that datapoints that are similar in feature space are probably of the same class. To determine which points are the closest a few different comparison functions are used [38]. In this thesis euclidean distance and cosine similarity has been used. The K value determines how many neighbors are allowed to vote in a classification and for this thesis values of 1, 10 and 100 has been used. One drawback of using K-nearest-neighbor is that every training datapoint has to be stored and for every classification the distance to every point has to be calculated. This makes the models large and slow for bigger datasets.

4
Methodology

This chapter explain the methods used to obtain the results in this thesis. First an overview of the algorithm and the steps taken to retrieve the data will be given. Secondly the methods for reconstruction, interpolating and time aligning of the data will explained. After that the each of the core parts of the algorithm will be explained. The core parts include the choice of similarity functions, the choice of machine learning algorithms and verification methods.

The first step to this project was to gather sensor data to perform the matching on. This was done using a set of hand annotated matched objects. No nonmatches were annotated so every matched object was given five nonmatches. To find nonmatches the five longest lived sensor objects that were found within the lifespan of the reference object were extracted and given the nonmatch label. If the matched sensor object was among the five extracted objects the label was changed to matching. If the matched sensor object was not extracted it was added to the dataset. When completed the dataset contained 4128 matching objects and 21625 nonmatching objects. The quality of the signals describing these objects varied so some reconstructing was necessary which is presented in section 4.1.

In figure 4.1 a flowchart of the steps taken in the matching algorithm are shown. The object signals are described in more detail in the setup section 2.2 of this report. The object signals are used to find a set of features through various similarity functions. The dimensionality of the features can be higher than what is suitable for the machine learning step and some may be uninformative so an option is to send these features through a dimensionality reduction function. After the optional mapping to a lower dimensionality the machine learning algorithms are used to classify the data.

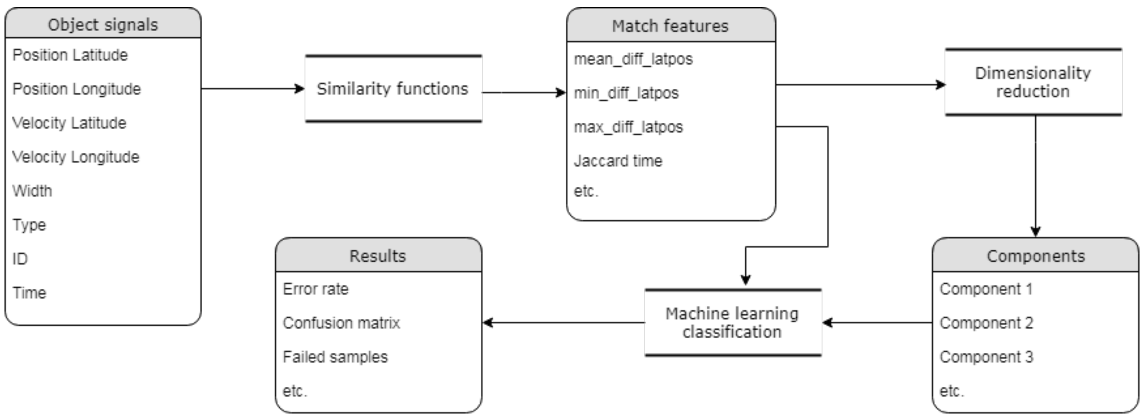


Figure 4.1: Flowchart of information through the matching algorithm.

4.1 Reconstructing and aligning

The raw measurements of the sensors have been processed to produce signals describing the objects detected but when the sensors do not provide adequate information the processing will assign these samples NaN values. When NaN values appear in short series an attempt to reconstruct the missing data can be made. This section will describe the methods used to reconstruct and align the data.

Both the reference and sensor system gives each object it finds an id. The range of numbers available as ids is limited and thus an id is reassigned to a new real world object after some time. When annotating the data the annotator marks two objects as matches which also generates a timestamp for the matching. In the cases where the same id has been used for multiple real objects the correct sequence of sampled data has to be selected to perform the matching on. In figure 4.2 there are 3 reference sequences with the same id which have been matched against a sensor sequence. A sequence has been defined as finished after a set number of NaN values occur in a row. This number has been set to 40 since it generated correct sequence splitting for all tested objects when testing on the dataset.

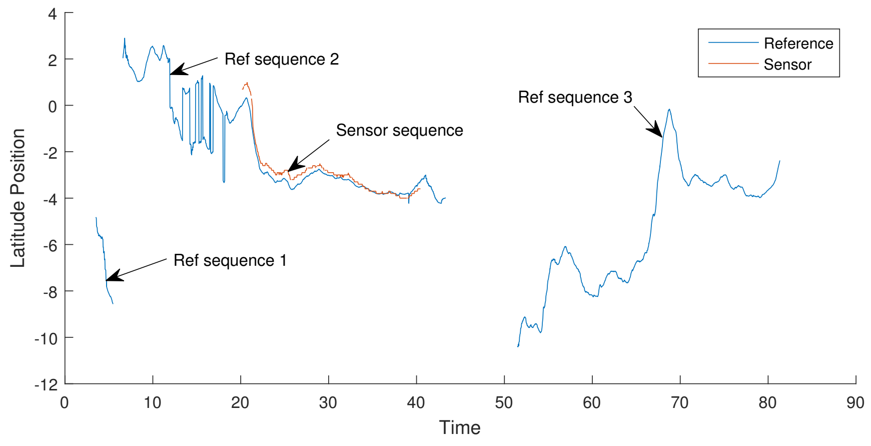


Figure 4.2: Sampled latitude position over time for a matching pair.

The sequences that best matches the annotated timestamp and has some time overlap are selected as the sequences to compare. In figure 4.3 the selected sequences from the the signals in figure 4.2 are shown. Within these sequences there can be short sections of NaN values due to flickering in the sensor signals. This is undesirable for the matching algorithm so missing samples are reconstructed using linear interpolation which is implemented as

$$p_{miss} = \frac{(t_{miss} - t_{prev})(p_{next} - p_{prev})}{t_{next} - t_{prev}} \tag{4.1}$$

where subscript miss denotes the missing sample, prev denotes the previous valid sample and next denotes the next valid sample.

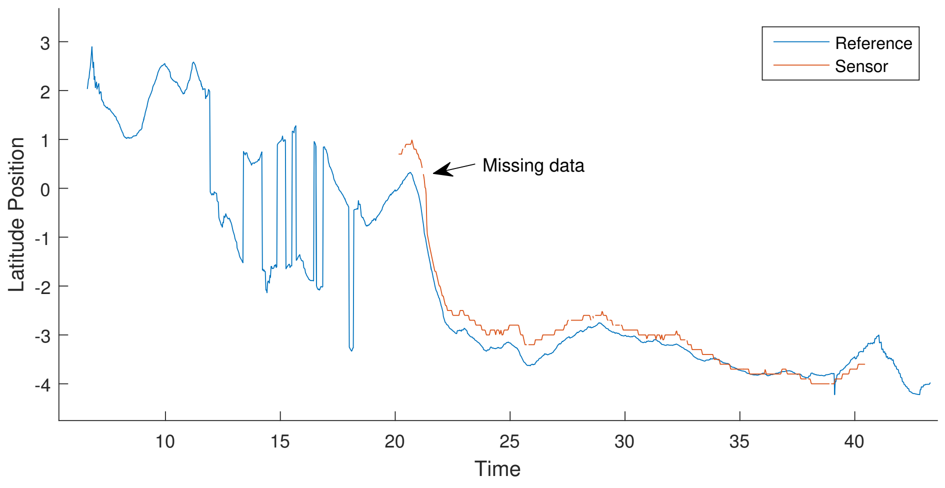


Figure 4.3: Showing correctly selected sequence and highlighting missing data.

Once the sequence has been repaired the samples can be used in similarity functions (described in section 4.2). In figure 4.4 the repaired sequences are shown and the time overlap has been highlighted. The overlap has been highlighted since it is the most important area of the sequence.

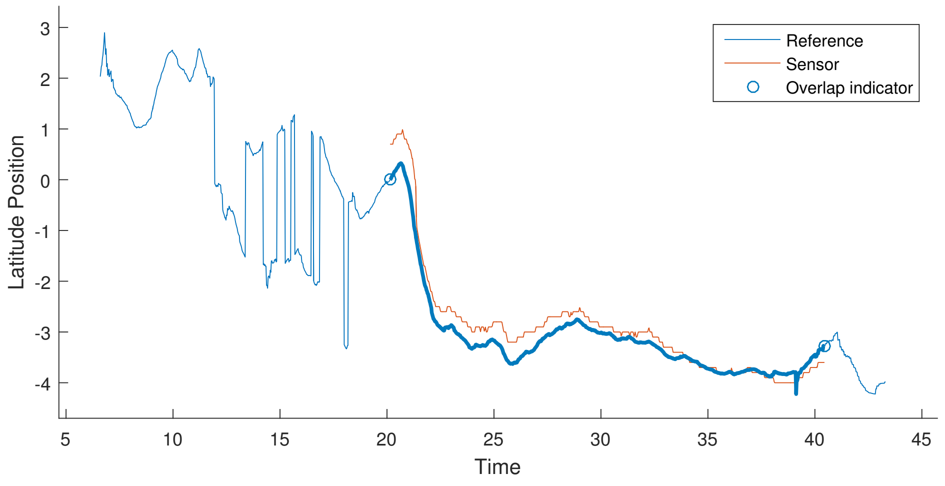


Figure 4.4: Showing the results of linear interpolation and highlighting the time overlap of the sensors.

The sensor and reference sequences in this section have been plotted over time to show the similarity of the sequences but since they are sampled at different rates they are not as similar in sample domain. To achieve the similarity seen in the figures in sample domain aswell the signals from the slower reference system are upsampled.Upsamling and synchronizing the reference signals is done using a clock that timestamps every sample of both systems. Since the time of measurement is known the samples from the slower system are placed at the corresponding time in the faster sample domain. The samples that does not have a corresponding sample are given values through linear interpolation as in equation 4.1.

4.2 Similarity functions

Similarity functions take the object signals from both the reference and sensor system and transform these signals into a feature. This section describes how and which similarity functions have been used in this project.

One similarity function that has been used to compare similarity of sets for instance in image segmentation is the Jaccard index [39]. In this project Jaccard index has been used to describe time overlap. The formula for Jaccard index in this context is

$$J(T_{r,i}, T_{s,j}) = \frac{|T_{r,i} \cap T_{s,j}|}{|T_{r,i} \cup T_{s,j}|} \tag{4.2}$$

where $T_{r,i}$ and $T_{s,j}$ are sets of sample times and $|T_{r,i}|$ denotes the number of elements in the set $T_{r,i}$.By letting $T_{r,i}$ be the time where object i is found by the reference system and $T_{s,j}$ be the time where object j is found by the sensor system. $J(T_{r,i}, T_{s,j})$ is then a relative measurement of how much time the two objects have been seen together.

The difference of signals is used for a couple of similarity functions

$$\Delta P_{lat} = P_{r,lat} - P_{s,lat} \tag{4.3}$$

where P is a column vector of all position values for the time samples $T_{r,i} \cap T_{s,j}$. The difference is used for a number of different similarity functions such as the max, min, mean and by considering the column vector P a vector in $\mathbb{R}^{|P|}$ vector comparisons such as norm of ΔP and cosine similarity of P_r and P_s can also be applied.

By squaring ΔP element-wise in latitude and longitude, then taking the element wise square root gives the euclidean distance in each point which is also used as a measurement for similarity. The formula

$$D = \sqrt{\Delta P_{lat} \odot \Delta P_{lat} + \Delta P_{\mathbf{lgt}} \odot \Delta P_{lgt}} \tag{4.4}$$

where $\odot$ denotes Hadamard product and the square root is taken element wise resulting in the euclidean distance in each sample from which the min, max, mean etc. can be taken to produce matching features. An alternative measure of distance that has been used is the Manhattan distance which is the sum of the difference in latitudinal and longitudinal direction.

Additional similarity functions such as applying the position methodology presented above to velocity measurements has been experimented with throughout the project. Another feature that was experimented with is the distance from the ego-vehicle to the reference object. The argument for this similarity function was that the sensor may behave differently for close objects compared to objects further away. Additionally the probabilities of the measurements as normal distributions has been experimented with such that an object pair has been rated based on how often the reference position is within the 3σ-level ellipse of the sensor position. Research suggests that adding features such as these additional similarity functions may hurt the algorithmic stability. [40] and experiments showed little gain in accuracy so these additional similarity functions were not included in the results presented in chapter 5.

4.3 Dimensionality reduction

In order to evaluate PCA we used the PCA function supplied with MATLABs *Statistics and Machine Learning Toolbox* [41].

kPCA was not applied to the whole dataset of 25753 samples, as a feasible solution could not be found for it. The restriction in kPCA is the eigenvalue decomposition of the centered kernel matrix. As the decomposition in equation 3.20 (or *eigs* command in algorithm 3) involves a time complexity of $\mathcal{O}(N^3)$ it became very time consuming and occupied large amounts of memory. Here N is the number of samples. If all 25753 samples would be in use, the kernel matrix $\mathbf{K}$ which is a square matrix with width and height equal to the number os samples would take up roughly 5GB of space.

The trivial solution was to use a subset of 5000 samples an algorithm implemented as described in 3. The only tested kernel funciton was a RBF with $\sigma = 1$ from table 3.1.

```
input  : data_in. Matrix of samples with number of samples equal to N.
input  : dim_out. Dimension of the output, ie. the number of principle
         components to be extracted.
output: data_out. Principle components, dim_out by number of samples.
for $row = 1$ to $N$ do
    for $col = 1$ to $row$ do
        $\mathbf{K}(row, col) \leftarrow \kappa(\text{row,col})$ ;    // construct kernel matrix
    end
end
$\mathbf{K} \leftarrow \mathbf{K} + \mathbf{K}^T$ ;    // as for loops only did half the matrix
for $row = 1$ to $N$ do
    $\mathbf{K}(row, row) \leftarrow \mathbf{K}(row, row)/2$ ;    // rescale the diagonal as it is
     added twice
end
$\widetilde{\mathbf{K}} = \mathbf{K} - \frac{1}{m}\mathbf{1}_m\mathbf{K} - \mathbf{K}\frac{1}{m}\mathbf{1}_m + \frac{1}{m^2}\mathbf{1}_m\mathbf{K}\mathbf{1}_m$ ;    // center the kernel matrix
$\mathbf{a}, \lambda \leftarrow eigs(\widetilde{\mathbf{K}})$ ;    // calculate eigenvectors and -values
$out\_index \leftarrow sort(\lambda)$ ;  // sort in descending order to export dim_out
 number of principle components.
for $component = 1$ to $dim\_out$ do
    $data\_out \leftarrow \mathbf{a}(out\_index(component)). * \widetilde{\mathbf{K}}$;    // fill data_out with
     most important component first and continue.
end
```

Algorithm 3: Pseudo code of kPCA implementation.

4.4 Implementation of machine learning

The machine learning algorithms take the outputs from the similarity functions and learn to classify objects as matching or not. This section will explain how the machine learning tools have been used in this project.

The machine learning algorithms have been implemented in MATLAB using the *Statistics and Machine Learning Toolbox* [41] which provide implementations of algorithms such as Support Vector Machines [11], Decision trees [12], k nearest neighbour [13] etc. Artificial neural networks [8] will also be implemented in MATLAB using the *Neural Network Toolbox* [42].

The data is presented to the machine learning algorithms as entries in a feature matrix. Each row represents a pair of objects, one from the reference and one from the sensor system. The pairs of objects are described by the outputs of the similarity functions. An example of how some samples are presented to the machine learning algorithms are show in table 4.1.

isMatch	ΔLatPos	ΔLgtPos	Euclidean	Manhattan	TimeOverlap
0	0,52	45,65	45,65	46,17	0,89
0	9,52	27,41	29,02	36,93	0,84
0	1,09	24,26	24,31	25,35	0,73
1	0,27	3,58	3,59	3,85	0,43

Table 4.1: Example of feature matrix entries as presented to the machine learning algorithms.

Decision trees have been used with the Gini impurity function and three different sizes of 4, 20 and 100. In bagging and boosting 30 trees of size 20 have been used. For the support vector machine linear, gaussian and polynomial kernels have been used. The order of the polynomial kernels was set to 2 and 3. In k nearest neighbour the euclidean and cubic distance in feature space was used to evaluate which neighbours were nearest. The neural networks were designed with a 2-layer layout and 10 hidden neurons in the hidden layer. The tanh activation functions was used.

4.5 Validation of machine learning algorithms

This section will describe the validation methods considered to evaluate the performance of different machine learning algorithms and present the decision process involved in choosing a method to evaluate performance within this thesis.

To evaluate the performance of an algorithm a method know as k-fold cross validation was used. In k-fold cross validation the dataset is divided into k randomly selected subsets of equal size, see figure 4.5.

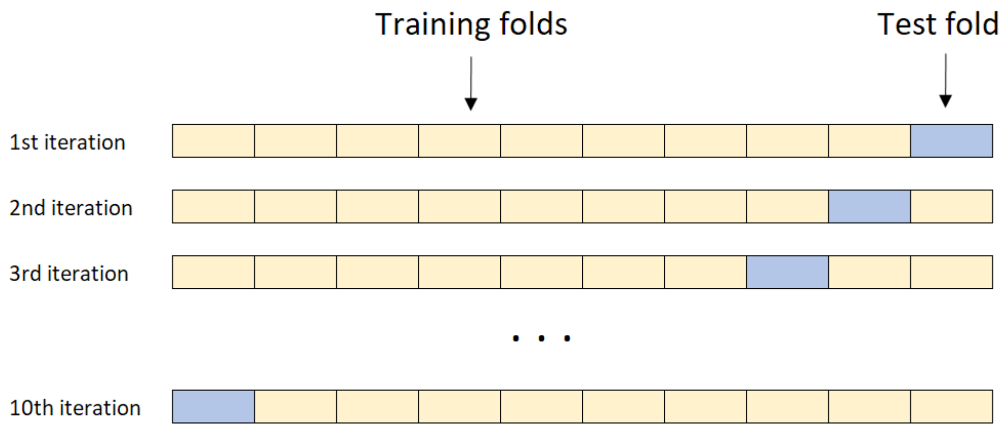


Figure 4.5: Example of selected training folds and test fold for each iteration. In this case k = 10, thus 10 folds and iterations are made.

Each subset acts as validation data once and training data k-1 times. In this thesis the value chosen for k was 10 which means every machine learning algorithm was trained and tested 10 times. A sample was given the classification it got from the machine learning algorithms when the sample was in the validation data. When all k models had made their predictions the results were compared to the labelled data and the algorithms were ranked based on their error rate and the type of errors made. The type of errors are presented using confusion matrices where classifying a pair as matching when it is not is considered worse than classifying a pair as nonmatching when they actually match. The type of error matters since the samples falsely classified as matching might be used to evaluate the performance of a sensor.

An alternative method for validation was the hold out validation method where a predefined subset of the data is selected for validation and the rest is used as training. The benefit of using k-fold cross validation over holdout validation is that there is no ambiguity in the choice of validation data and results from k-fold cross validation tend to better describe the performance of the algorithm [43].

4.6 Algorithmic stability

This section will describe how a trained model was tested for algorithmic stability. The notion of algorithmic stability is used in this thesis to describe the generalization abilities of the model as it is used to classify unseen data with added noise. The added noise is supposed to emulate the change of sensor, say from manufacturer A to manufacturer B's counterpart. Different noise models will also give insight in how model degradation occur. To explore the algorithmic stability the models shown in figure 4.6 was used. Noise model 1 is represented by a third degree polynomial, created in collaboration with a test engineer at Volvo to give it characteristics and scaling that closely correspond to a real example. Model 2, 3 and 4 (2 and 3 are polynomial, 4 is sigmoid) are similar functions with exaggerated features with the purpose to further deteriorate the models. The noise generation works by letting the noise models generate a relative error depending on how far away the object is. The relative error times the current distance generate a standard deviation that can be used in an Gaussian distribution with mean $\mu = 0$ $\mathcal{N}(\mu, \sigma^2)$ to generate a suitable additive noise.

For example, if an longitudinal noise shall be added with model 1 and the object is 80m away, a standard deviation of 10% of the current distance will be used to draw a random number from the a Gaussian probability density function. $\sigma = 0.1 \cdot 80 = 8$, so a noise from the distribution $\mathcal{N}(\mu, 8^2)$ is added.

The algorithmic stability is checked by the combination of k-fold validation as described in section 4.5 and data augmentation with different noise models, figure 4.6.

The combination works as follows:

1. Divide the data in 10 subsets.
2. Train a classifier 9 of the subsets and leave one part for testing.
3. Process the testing subset in the noise model.
4. Classify the subset with the added noise.
5. Store results and repeat from step 2 but with a different subset for training and testing.

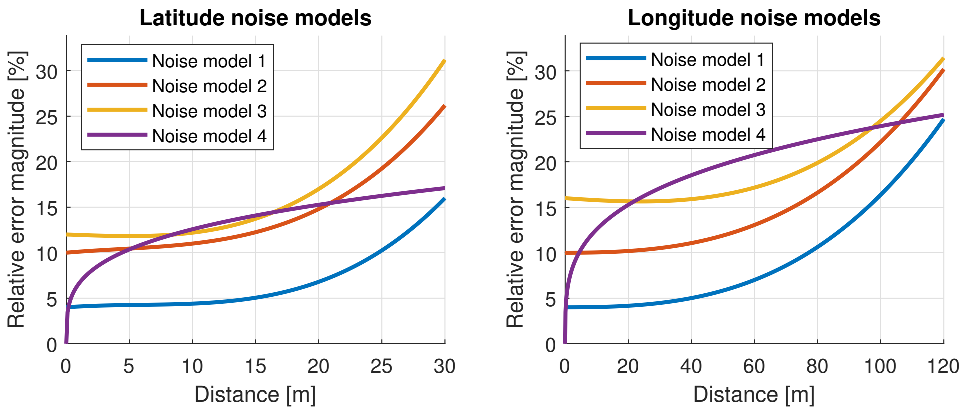


Figure 4.6: Models used to generate noise that when added to the sensor, emulate another manufacturer's sensor.

4.7 Experimental motivation for the use of similarity functions

This section describes an alternative methodology where the similarity functions have been replaced by a larger machine learning model which takes sampled time-series as input and returns a classification.

The model takes the sampled positions of object pairs as four timeseries, the two first are latitude and longitude position from the sensor and the second two are latitude and longitude position from the reference. The motivation for this setup was that since the similarity functions aim to produce a descriptive scalar value from sampled series some dynamics within the series might be lost. A model such as Long Short Term Memory (LSTM) neural networks that can handle historical information might capture information lost in the similarity functions. This type of setup would also remove the engineering work required to find suitable similarity functions.

A test was performed using a network with a LSTM layer with 100 nodes followed by a fully connected layer with 2 nodes followed by a softmax and finally a classification layer. The model was evaluated using 5-fold cross validation.

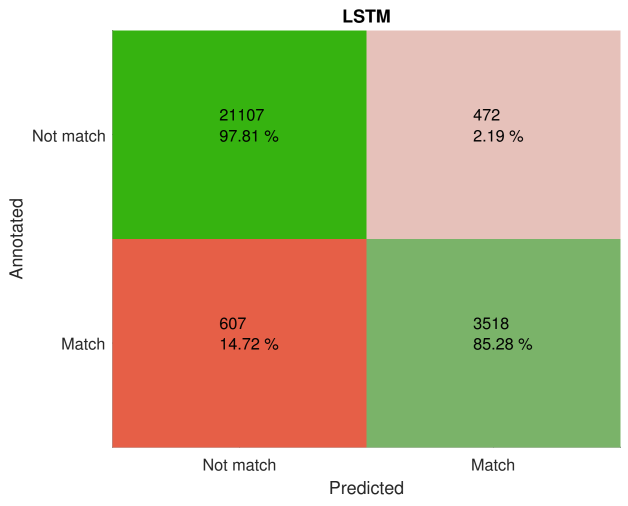


Figure 4.7: Confusion matrix of the results running LSTM network.

In figure 4.7 the results from the LSTM classification can be found. Only 85.28% of the matching object pairs were correctly classified which is not on par with the results achieved using similarity functions, shown in chapter 5. Since this experiment did not show promising results LSTM networks were not further investigated as a solution in this thesis.

5
Results

This chapter presents the results of the thesis. The results include effects of using dimensionality reduction, performance of different machine learning (ML) algorithms and results from noise rejection tests.

5.1 Dimensionality and noise reduction

The procedures of PCA and kPCA described in section 3.1 have a record of improving the outcome of classifiers in certain cases [44], [45]. All results and information about the different configurations can be found in appendix B for both PCA and kPCA.

In short, the results told that PCA and kPCA had in general a negative effect across the board.

5.1.1 PCA

When PCA was applied in conjunction with machine learning to classify the data in the set RaCam2Lidar, it showed a few improvements but mostly a tendency to decrease the performance of the classification. On the Cam2Lidar dataset, the PCA decreased the performance in all cases except one where there were no difference. In table 5.1 and 5.2 the cases that received a positive or neutral effect from PCA on the overall accuracy can be seen.

In table 5.1 the best combiation of ML and PCA was the medium tree with a dimensionality reduction from 5 to 4 ending up with an overall accuracy of 99.40%. A boosted tree without PCA had an overall accuracy of 99.44%, which is the highest accuracy of all tested algorithms for the RaCam2Lidar dataset.

Dataset	Algorithm	Dim. reduction	w/o PCA	w/ PCA	Change
RaCam2Lidar	KNN Cubic	5 to 2	99.29%	99.30%	0.02%
RaCam2Lidar	KNN Cubic	5 to 3	99.29%	99.30%	0.02%
RaCam2Lidar	Tree Medium	5 to 4	99.39%	99.40%	0.00%
RaCam2Lidar	Boosted Trees	-	99.44%	-	-

Table 5.1: Table showing where PCA had positive or neutral effect on the overall accuracy for the algorithm that ran on the RaCam2Lidar dataset. The overall accuracy is presented under the column w/o PCA and w/ PCA.

When it came to the Cam2Lidar dataset, see table 5.2, the only combination of ML and PCA that did not have a negative effect was a Gaussian SVM. Without a performance increase it scored even with a boosted tree and a Gaussian SVM without PCA.

Dataset	Algorithm	Dim. reduction	w/o PCA	w/ PCA	Change
Cam2Lidar	SVM RBF	6 to 5	98.09%	98.09%	0.00%
Cam2Lidar	Bagged Trees	-	98.09%	-	-
Cam2Lidar	SVM RBF.	-	99.09%	-	-

Table 5.2: Table showing where PCA had positive or neutral effect on the overall accuracy for the algorithm that ran on the Cam2Lidar dataset. The overall accuracy is presented under the column w/o PCA and w/ PCA.

In the instances where the algorithm improved in performance, the combination of PCA and ML never performed better then the best performing algorithm without PCA.

5.1.2 kPCA

When kPCA was applied in conjunction with machine learning to classify the data in the set RaCam2Lidar it did show a stable increase in performance when working with cosine kNNs but mostly a tendency to decrease the performance of the classification. On the Cam2Lidar dataset, the kPCA decreased the performance in all cases except when working with cosine kNNs. In table 5.3 and 5.4 the cases that received a positive effect from kPCA on the overall accuracy can be seen.

Dataset	Algorithm	Dim. reduction	w/o kPCA	w/ kPCA	Change
RaCam2Lidar	SVM Cubic	5 to 2	99.02%	99.20%	0.18%
RaCam2Lidar	kNN Cosine	5 to 2	97.36%	99.44%	2.14%
RaCam2Lidar	SVM Cubic	5 to 3	99.02%	99.40%	0.38%
RaCam2Lidar	SVM Fine G.	5 to 3	99.44%	99.52	0.08 %
RaCam2Lidar	kNN Cosine	5 to 3	97.36%	99.48	2.18 %
RaCam2Lidar	SVM Cubic	5 to 4	99.02%	99.40	0.38 %
RaCam2Lidar	SVM Fine G.	5 to 4	99.44%	99.48	0.04 %
RaCam2Lidar	kNN Cosine	5 to 4	97.36%	99.52	2.14 %
RaCam2Lidar	SMV Cubic	5 to 5	99.02%	99.44	0.42 %
RaCam2Lidar	kNN Cosine	5 to 5	97.36%	99.46	2.16 %
RaCam2Lidar	Boosted Trees	-	99.58%	-	-

Table 5.3: Table showing where kPCA had positive effect on the overall accuracy for the algorithm that ran on the RaCam2Lidar dataset. The overall accuracy is presented under the column w/o PCA and w/ PCA.

When it came to the Cam2Lidar dataset, see table 5.4, the only combination of ML and kPCA that did not have a negative change was a fine Gaussian SVM. Without a performance increase it scored even with a boosted tree and a fine Gaussian SVM without PCA.

Dataset	Algorithm	Dim. reduction	w/o kPCA	w/ kPCA	Change
Cam2Lidar	KNN Cosine	6 to 2	98.26%	98.48%	0.22%
Cam2Lidar	KNN Cosine	6 to 3	98.26%	98.44%	0.18%
Cam2Lidar	KNN Cosine	6 to 4	98.26%	98.88%	0.63%
Cam2Lidar	KNN Cosine	6 to 5	98.26%	98.82%	0.57%
Cam2Lidar	KNN Cosine	6 to 6	98.26%	99.06%	0.81%
Cam2Lidar	Bagged Trees	-	99.54%	-	-

Table 5.4: Table showing where kPCA had positive effect on the overall accuracy for the algorithm that ran on the Cam2Lidar dataset. The overall accuracy is presented under the column w/o PCA and w/ PCA

As in the case of PCA, kPCA did not enter amongst the top tier classifiers show any improvement for the classifiers, thus ending any further investigation into improving the speed and increasing the sample size of kPCA.

5.2 Performance on different sensor systems

This section will present results achieved when testing the same algorithms on the three different dataset RaCam2Lidar, Cam2Lidar and Radar2Lidar. The features used in this section are ΔLatPos, ΔLgtPos, Euclidean, Manhattan, TimeOverlap.

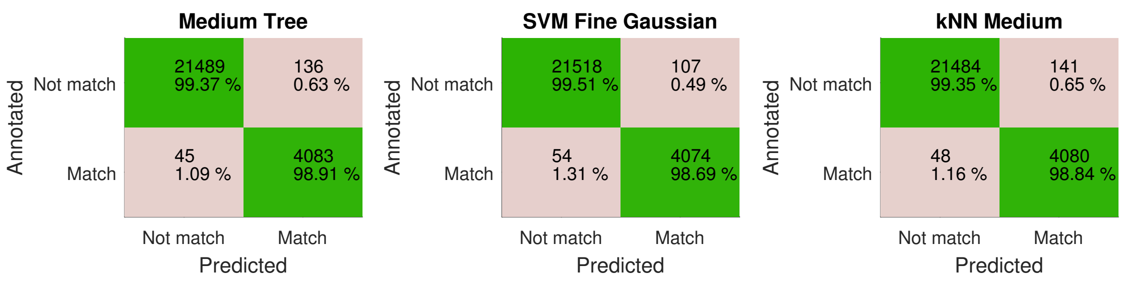


Figure 5.1: Performance confusion matrix of three different algorithms on the RaCam2Lidar dataset.

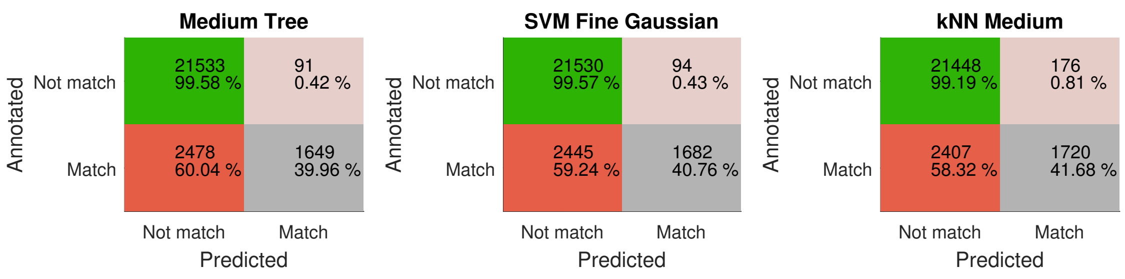


Figure 5.2: Performance confusion matrix of three different algorithms on the Radar2Lidar dataset.

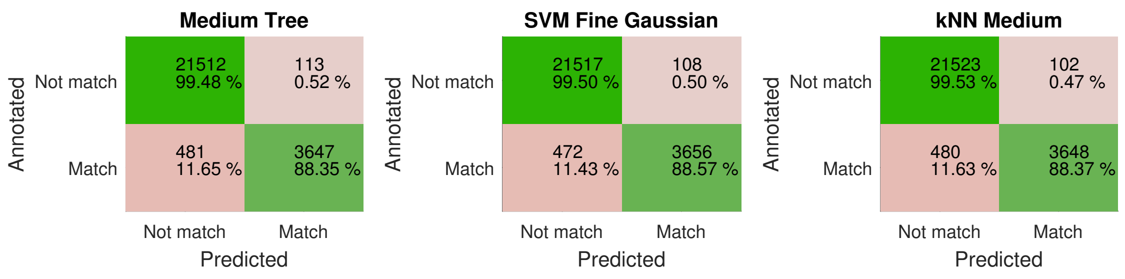


Figure 5.3: Performance confusion matrix of three different algorithms on the Cam2Lidar dataset.

Comparing the results when matching on RaCam2Lidar (figure 5.1) data to both Cam2Lidar (figure 5.2) and Radar2Lidar (figure 5.3) data one can see a decrease in accuracy. Matching on radar measurements alone gives results that are only slightly better than using an always false strategy. All algorithms do however benefit from the information gained by fusing the radar measurements with the camera measurements.

The medium tree used is a tree with 20 splits using the Gini impurity function, SVM Fine Gaussian uses a Gaussian kernel with a scale of 0.6 and kNN Medium uses the 10 nearest neighbours with euclidean distance as it's cost functions.

5.3 Boosting and bagging the classifiers

By applying techniques such as boosting and bagging to the learning algorithms the performance may be increased. This section presents the results and performance difference when applying these techniques on decision trees. Decision trees are used since they provided best results in the previous tests and are fast to train.

Dataset	Algorithm	Overall accuracy	Change
RaCam2Lidar	Tree Medium	99.37%	-
RaCam2Lidar	Boosted Trees	99.41%	0.04%
RaCam2Lidar	Bagged Trees	99.29%	-0.08%
Cam2Lidar	Tree Medium	97.69%	-
Cam2Lidar	Boosted Trees	97.76%	0.07%
Cam2Lidar	Bagged Trees	97.62%	-0.07%
Radar2Lidar	Tree Medium	90.02%	-
Radar2Lidar	Boosted Trees	90.07%	0.05%
Radar2Lidar	Bagged Trees	90.13%	0.11%

Table 5.5: Table showing effects of bagging and boosting on decision trees.

In table 5.5 one can see that boosting increased the performance in all three cases while bagging decreased the performance for two of the dataset but increased the performance for the Radar2Lidar case. Since the RaCam data is used when the car makes decisions the results from this dataset are more important.

5.4 Algorithmic stability

By adding four different types of noise to the unseen data as described in section 4.6, we can track how the different models react to noise. Table 5.6 show the change for the different algorithms and noise models. This table show that the overall accuracy rate decreases with 0.03 - 5.35%. The true positive rate decreases with 0 - 34.08%. All results and information about the different configurations can be found in appendix C.

Figure 5.4 and 5.5 show that there is a clear difference between the models. Decision trees and boosted trees still perform with a true positive rate above 95% but kNN and SVMs fall below 95% after noise model 2.

Algorithm	w/o Noise model	1	2	3	4
Model degradation with respect to overall accuracy					
Tree Medium	99.39%	-0.03%	-0.23%	-0.55%	-1.34%
SVM Quadratic	99.06%	-0.14%	-0.74%	-3.17%	-5.17%
SVM Fine G.	99.32%	-0.11%	-0.70%	-2.85%	-5.35%
KNN Cubic	99.29%	-0.07%	-0.55%	-2.26%	-4.36%
Boosted Trees	99.44%	-0.23%	-0.43%	-0.79%	-0.92%
Model degradation with respect to true positive					
Tree Medium	99.39%	0.00%	-0.68%	-2.52%	-7.91%
SVM Quadratic	99.06%	-0.19%	-4.31%	-20.45%	-33.04%
SVM Fine G.	99.32%	-0.78%	-4.65%	-18.36%	-34.08%
KNN Cubic	99.29%	-0.56%	-3.68%	-14.61%	-28.08%
Boosted Trees	99.44%	-0.94%	-2.03%	-4.26%	-5.35%

Table 5.6: Table of performance deterioration for different models. The second column (w/o Noise model) show the overall accuracy (upper) or true positive rate (lower) when no extra noise has been added. The third to sixth column show the rate of change from without noise model to the noise model corresponding to the number in the column.

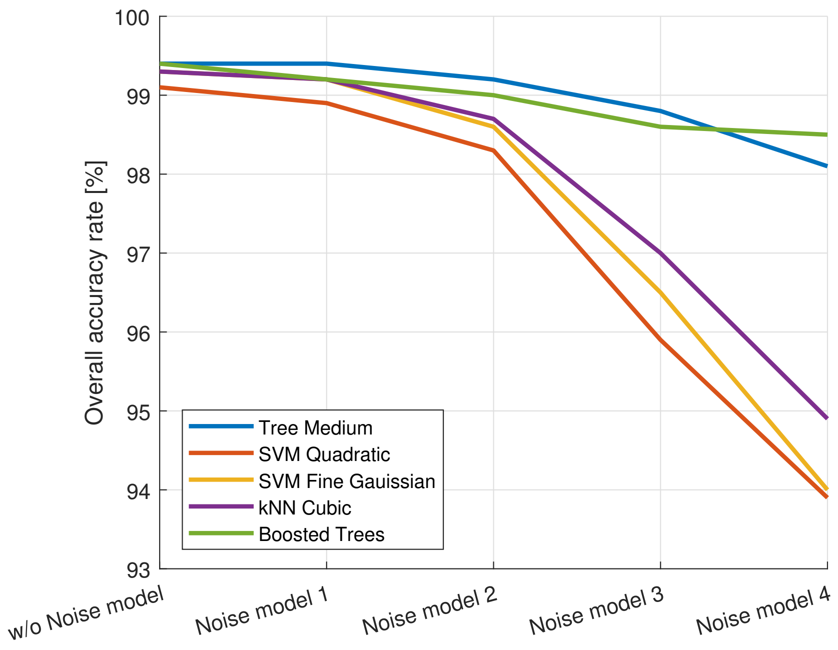


Figure 5.4: Degradation of overall accuracy when 4 different noise models is applied to the sensor data.

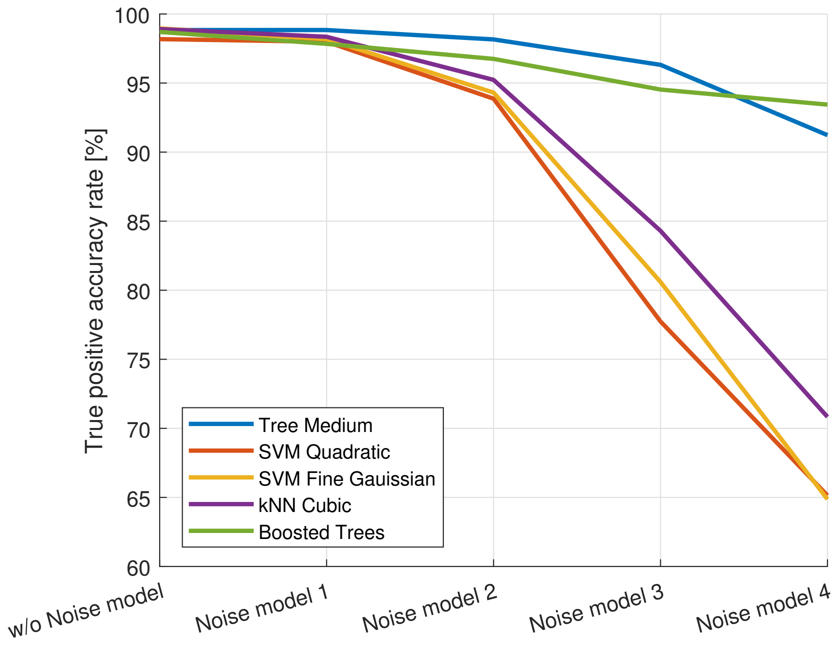


Figure 5.5: Degradation of true positive when 4 different noise models is applied to the sensor data.

6 Discussion

This chapter will provide discussions about the results presented and methodology used in this thesis. Each section covers a subject the authors found interesting to discuss and they are written to be read in any order.

6.1 Extraction of samples

The matching object pairs used in this thesis were, as mentioned in the preliminaries, annotated prior to the initialization of this thesis. Some of the faulty classifications that the classifiers makes are due to incorrect annotations or ambiguous cases. The original annotation of these cases has been kept since it was decided that the judgment of the original annotator should influence the data rather than the judgment of the authors to guarantee that the samples were not changed to agree with the algorithm and thus exaggerate the results presented in this thesis.

The nonmatching object pairs were, as mentioned in the methodology section, found by taking the longest lived vehicle objects with some time overlap. The argument for this methodology was that objects that are detected for a longer period often correspond to some real world object. The objective of this thesis was to match object pairs to each other and by choosing the longest lived objects classified as vehicles the dataset was not flooded by false detections i.e sensor noise.

One issue with this methodology is that in some cases a detected object, such as a car, might change id throughout its lifespan. For example if a car temporarily exit the sensor's line of sight (without leaving the reference's line of sight) and then reappear it will be given a new id, see figure 6.1. The annotator might not notice the new id and thus a matching pair might be falsely considered nonmatching and would contribute to a higher false positive rate. These issues have not been compensated for in the dataset because they capture an unwanted behaviour of the sensor and correcting them would require a person to go through every entry in the dataset which was considered too time-consuming for this project.

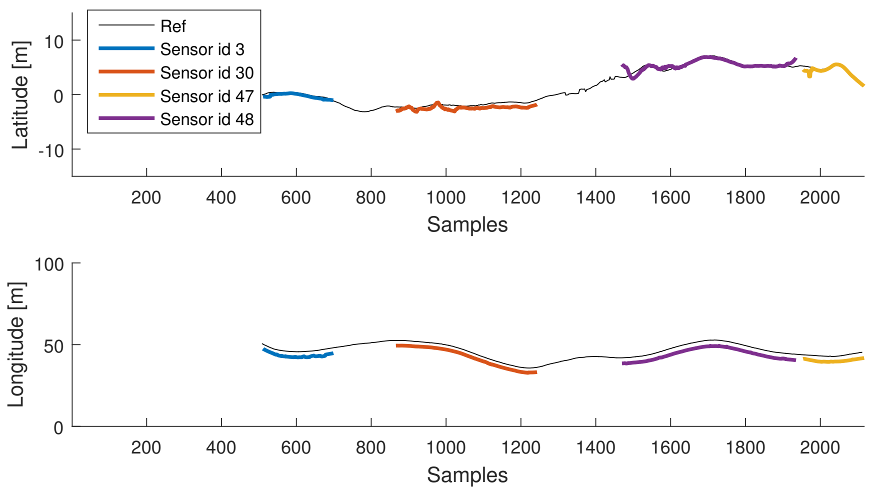


Figure 6.1: Example from RaCam2Lidar dataset which highlights the issue about new ids for a sensor object. As an object leaves the system's line of sight, it will be assigned a new id when reappearing. A problem exist if the annotator don't annotate all four ids as a match to the reference.

6.2 Different machine learning methods

Boosted trees have shown the best results so for pure accuracy the machine learning algorithm of choice is the boosted trees. The boosting increased the performance of decision trees on the RaCam2Lidar dataset by 0.04% which is an increase that comes at a cost. The boosting was performed with 30 trees which means 30 times as much training is required. On the dataset in this thesis that was not an issue but boosting also makes analyzing the errors more complex. If a single tree would be used one could easily traverse the tree by hand to see what feature caused a miss-classification. This is not as easy with the boosted trees since one would have to traverse 30 trees and then look at the weights corresponding to each tree to find out why and where the algorithm made the crucial mistake. If 0.04% increase in accuracy is worth more than 30 times the work and losing the interpretability is a design choice.

When analyzing the results of the support vector machines on all datasets they perform almost on par with the trees but since they do not outperform them and are more time-consuming to train, harder to analyze and generalizes worse to noisy data. The summary is that they show no real benefit over the decision trees.

The kNN algorithm show classification abilities on par with the decision trees but since the kNN model saves all datapoints, which with a dataset of 25753 entries and a dimensionality around 5 the space complexity of such a model is much larger than the complexity of a decision tree. As a comparison a tree with size 20 which was used in section 5.2 only needs to store 20 feature-value pairs in a binary tree structure.

6.3 Similarity functions

The task of the similarity functions was to describe how similar a reference and a sensor object are as a scalar value. In this work the main source of information has been the relative position to the ego-vehicle. Expanding the amount of similarity functions could possibly improve the accuracy of the algorithm but since research suggests that such expanding comes at a cost of reduced algorithmic stability [40] and given that five features resulted in 99.4% correct classifications the amount of similarity functions were not increased further.

The similarity functions return scalar values describing time-series which means some information is lost when calculating the output. In section 4.7 an attempt to classify matches based on sequences of data was made. Since the LSTM network presented had access to more information one might suspect that it would produce better classifications but it turns out this strategy was not very effective. The underwhelming results are probably due to the difficulty to train neural networks properly and perhaps lack of data. Disciplines where LSTM networks are used successfully include translation tasks [46] where the amount of available data is far beyond what was available in this thesis. The network architecture used may have been too naively designed but since the initial testing showed poor results investigating other methods such as SVM and decision trees rather than configuring the network was decided to be the better option.

6.4 One to one or many to one classifications

In a traffic scenario there are often many different detected objects present at any given time and one strategy could be to assume that each reference object should be matched to the most likely object among the objects found by the sensor. Using such a strategy would try to answer the question *Which sensor object corresponds to this reference object?*. This is what is meant by many to one classification.

Alternatively one could pick a pair of objects, one from each system, and compare them to answer the question *Do these two measured objects describe the same physical object?*. This is what is meant by one to one classification.

In this thesis the designed algorithm answers the second question which introduces a few errors in cases where two different objects found by the sensor can be matched to a single reference object. This type of error tends to happen when driving behind a truck transporting other vehicles since the reference system usually consider truck and cargo as one object while the sensor separate the transporting and transported vehicles. In figure 6.2 a truck is carrying tractor. The reference has considered the truck and the cargo as one object while the sensor has classified them as two separate vehicles. In this case the algorithm classify both the truck and the cargo as matching the reference object but the annotation says that only the tractor should be considered a match. This type of error can be considered acceptable since it has more to do with how the two systems differentiate between different objects rather than the actual matching.

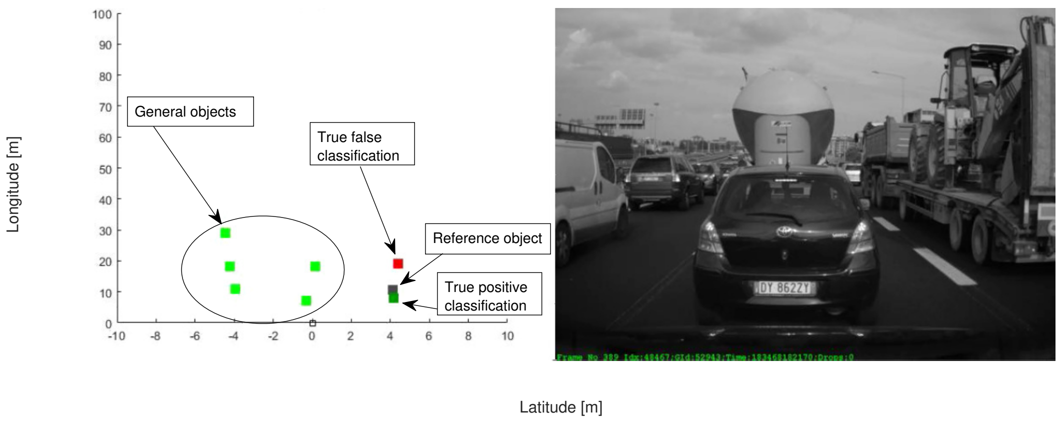


Figure 6.2: The left plot show identified objects in a top view. The right image correspond to the current frame from the camera.

The first alternative introduces other problems such as guaranteeing that every reference object has a detected sensor object to match against. This is not always the case since the reference system is located on the roof of the car and the sensor is placed in the windshield which means that the sensors field of view might be blocked by an obstacle while the reference can see over it.

7 Conclusion

This chapter will present a conclusion to the thesis project. It consists of a selected machine learning algorithm and a suggested implementation of how to use these findings to automate the verification of sensor performance.

We have concluded that it is possible to classify whether two sources describe the same object or not with an overall accuracy of $\sim 99\%$. According to the experiments a good choice of machine learning strategy is to use decision trees with boosting. Finding few informative features rather than many semi-informative has shown better results which also makes dimensionality reduction with PCA unnecessary.

A classifier such as the one above could be used to find biases in a system compared to the chosen reference system. Thanks to the low degradation rate of a tree for example, one could perform sensor analysis on several different sensors with the same model.

Future work

The research questions asked is answered which lead to scalability related questions and implementation. In terms of scalability it would be of interest to see how many types of objects can be incorporated into the classification; pedestrians, signs, road edges etc.

The matching performance found in this thesis shows that using boosted trees to match object pairs for automatic sensor verification could be an effective verification method. The future work is then to integrate the matching algorithm found here in the existing verification toolchain at Volvo.

Bibliography

[1] D. Sleet D. Mohan A. A. Hyder E. Jarawan C. Mathers M. Peden, R. Scurfield. World report on road traffic injury prevention. In *World Health Organization Geneva Tech. Rep.*, 2004.

[2] S. Singh. Critical reasons for crashes investigated in the national motor vehicle crash causation survey. In *National Highway Traffic Safety Administration (NHTSA) Washington DC Tech*, Feb 2015.

[3] J. Florbäck, L. Tornberg, and N. Mohammadiha. Offline object matching and evaluation process for verification of autonomous driving. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 107–112, Nov 2016.

[4] D. M. Gavrila and V. Philomin. Real-time object detection for ldquo;smart rdquo; vehicles. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 87–93 vol.1, 1999.

[5] Nick Hillier Julian Ryde. Performance of laser and radar ranging devices in adverse environmental conditions. In *Journal of Field Robotics*, volume 26, page 712–727, Sep 2009.

[6] J. Hasch, E. Topak, R. Schnabel, T. Zwick, R. Weigel, and C. Waldschmidt. Millimeter-wave technology for automotive radar sensors in the 77 ghz frequency band. *IEEE Transactions on Microwave Theory and Techniques*, 60(3):845–860, March 2012.

[7] Radar, camera, lidar and v2x for autonomous cars. `https://blog.nxp.com/automotive/radar-camera-and-lidar-for-autonomous-cars`. Accessed: 2018-01-12.

[8] J. J. Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, Sept 1988.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, June 2016.

[10] S. P. Singh, A. Kumar, H. Darbari, L. Singh, A. Rastogi, and S. Jain. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167, July 2017.

[11] Alexandre Kowalcyk. Support vector machines succinctly. In *Support Vector Machines Succinctly*, Oct 2017.

[12] L. Deng and J. Y. Song. Decision tree classification algorithm based on cost and benefit dual-sensitive. In *IEEE International Conference on Electro/Information Technology*, pages 320–323, June 2014.

[13] D. Jamma, O. Ahmed, S. Areibi, G. Grewal, and N. Molloy. Design exploration of asip architectures for the k-nearest neighbor machine-learning algorithm. In *2016 28th International Conference on Microelectronics (ICM)*, pages 57–60, Dec 2016.

[14] Klaus-Jürgen Bathe. *Finite element procedures.* Klaus-Jurgen Bathe, 2006.

[15] Rasmus Elsborg Madsen, Lars Kai Hansen, and Ole Winther. Singular value decomposition and principal component analysis. *Neural Networks*, 1:1–5, 2004.

[16] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural computation*, 10(5):1299–1319, 1998.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[18] CHRISTOPHER J.C. BURGES. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery 2*, pages 121–167, 1998.

[19] C. Schmid and L.T. Biegler. Quadratic programming methods for reduced hessian sqp. *Computers & Chemical Engineering*, 18(9):817 – 832, 1994. An International Journal of Computer Applications in Chemical Engineering.

[20] Wei-Lun Chao. A tutorial for support vector machine. `http://disp.ee.ntu.edu.tw/~pujols/Support%20Vector%20Machine.pdf`. Accessed: 2018-02-02.

[21] Matthew Bernstein. The radial basis function kernel. `http://pages.cs.wisc.edu/~matthewb/pages/notes/pdf/svms/RBFKernel.pdf`. Accessed: 2018-02-05.

[22] Cynthia Rudin. Kernels mit 15.097 course notes. *(Massachusetts Institute of Technology: MIT OpenCouseWare), http://ocw.mit.edu (Accessed [2018-02-26]). License: Creative Commons BY-NC-SA*, Spring 2012.

[23] Colin Campbell. An introduction to kernel methods. *Studies in Fuzziness and Soft Computing*, 66:155–192, 2001.

[24] Mathworks documentation. `https://se.mathworks.com/help/matlab/ref/tanh.html`. Accessed: 2018-02-07.

[25] Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines vinod nair.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[27] Course material for cs231n at standford uni. `http://cs231n.github.io/neural-networks-1/`. Accessed: 2018-02-08.

[28] R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989.

[29] Michael A.Nielsen. *Neural Networks and Deep Learning.* Determination Press, 2015.

[30] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1139–III–1147. JMLR.org, 2013.

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*, 2010.

[32] Laura Elena Raileanu and Kilian Stoffel. Theoretical comparison between the gini index and information gain criteria. *Annals of Mathematics and Artificial Intelligence*, 41(1):77–93, May 2004.

[33] Lecture notes from the course cse 802 pattern recognition and analysis at michigan state university. `http://www.cse.msu.edu/~cse802/DecisionTrees.pdf`. Accessed: 2018-03-27.

[34] Tapio Elomaa and Matti Kääriäinen. An analysis of reduced error pruning. *CoRR*, abs/1106.0668, 2011.

[35] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 55(1):119–139, August 1997.

[36] X. Liu, Y. Dai, Y. Zhang, Q. Yuan, and L. Zhao. A preprocessing method of adaboost for mislabeled data classification. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 2738–2742, May 2017.

[37] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, Aug 2000.

[38] J. Laaksonen and E. Oja. Classification with learning k-nearest neighbors. In *Neural Networks, 1996., IEEE International Conference on*, volume 3, pages 1480–1483 vol.3, Jun 1996.

[39] R. Shi, K. N. Ngan, and S. Li. Jaccard index compensation for object segmentation evaluation. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 4457–4461, Oct 2014.

[40] H. Xu, C. Caramanis, and S. Mannor. Sparse algorithms are not stable: A no-free-lunch theorem. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):187–193, Jan 2012.

[41] Statistics and machine learning toolbox. `https://se.mathworks.com/products/statistics.html`. Accessed: 2018-01-11.

[42] Neural network toolbox. `https://se.mathworks.com/help/nnet/index.html`. Accessed: 2018-04-09.

[43] S. Yadav and S. Shukla. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pages 78–83, Feb 2016.

[44] S. Xie and S. Krishnan. Signal decomposition by multi-scale pca and its applications to long-term eeg signal classification. In *The 2011 IEEE/ICME International Conference on Complex Medical Engineering*, pages 532–537, May 2011.

[45] M. S. Reza and J. Ma. Ica and pca integrated feature extraction for classification. In *2016 IEEE 13th International Conference on Signal Processing (ICSP)*, pages 1083–1088, Nov 2016.

[46] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *CoRR*, abs/1708.02709, 2017.

A
Proof for reordering matrices

Showning that

$$\phi(x)\phi(x)^T w = \{\phi(x) \cdot w\}\phi(x)^T \tag{A.1}$$

where $\phi(x) = [\phi(x_1), \ldots, \phi(x_i)]^T$ and $w = [w_1, \ldots, w_i]^T$

$$\begin{aligned}
\phi(x)\phi(x)^T w &= \begin{bmatrix} \phi(x_1)\phi(x_1) & \phi(x_1)\phi(x_2) & \ldots & \phi(x_1)\phi(x_i) \\ \phi(x_2)\phi(x_1) & \phi(x_2)\phi(x_2) & \ldots & \phi(x_2)\phi(x_i) \\ \vdots & \vdots & \ddots & \vdots \\ \phi(x_i)\phi(x_1) & \phi(x_i)\phi(x_2) & \ldots & \phi(x_i)\phi(x_i) \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_i \end{bmatrix} && \text{(A.2)} \\
&= \begin{bmatrix} \phi(x_1)\phi(x_1)w_1 + \phi(x_1)\phi(x_2)w_2 + \cdots + \phi(x_1)\phi(x_i)w_i \\ \phi(x_2)\phi(x_1)w_1 + \phi(x_2)\phi(x_2)w_2 + \cdots + \phi(x_2)\phi(x_i)w_i \\ \vdots \\ \phi(x_i)\phi(x_1)w_1 + \phi(x_i)\phi(x_2)w_2 + \cdots + \phi(x_i)\phi(x_i)w_i \end{bmatrix} && \text{(A.3)} \\
&= \begin{bmatrix} \{\phi(x_1)w_1 + \phi(x_2)w_2 + \cdots + \phi(x_i)w_i\}\phi(x_1) \\ \{\phi(x_1)w_1 + \phi(x_2)w_2 + \cdots + \phi(x_i)w_i\}\phi(x_2) \\ \vdots \\ \{\phi(x_1)w_1 + \phi(x_2)w_2 + \cdots + \phi(x_i)w_i\}\phi(x_i) \end{bmatrix} && \text{(A.4)} \\
&= \begin{bmatrix} \phi(x_1)w_1 + \phi(x_2)w_2 + \cdots + \phi(x_i)w_i \end{bmatrix} \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_i) \end{bmatrix} && \text{(A.5)} \\
&= \{\phi(x) \cdot w\}\phi(x)^T && \text{(A.6)}
\end{aligned}$$

B Dimensionality and noise reduction tables.

RaCam2Lidar. Data splitted for first 25753 out of 27373 and validated with 10-fold crossvalidation

Setup						RESULT												
Date	Information	Comment	ML Algorithm		Features	Accuracy	TP	FP	TN	FN	Sum	Accuracy	TP	FP	TN	FN	Algorithm	Change
26-03-2018	Splits = 20. Ginis diversity. Surrogate off	PCA no dim red	Tree Medium	5->5	lat lgn euc, manh	99.383%	4068	99	21526	60	25753	99.383%	98.55%	0.46%	99.54%	1.45%	PCA 5 Tree Medium	-0.01%
26-03-2018	Kernel scale = 0.35	PCA no dim red	SVM Fine G	5->5	lat lgn euc, manh	99.251%	4084	149	21476	44	25753	99.251%	98.93%	0.69%	99.31%	1.07%	PCA 5 SVM Fine G	-0.07%
26-03-2018	Nr neighbors = 10. Cubic. Equal	PCA no dim red	KNN Cubic	5->5	lat lgn euc, manh	98.792%	4064	247	21378	64	25753	98.792%	98.45%	1.14%	98.86%	1.55%	PCA 5 KNN Cubic	-0.50%
26-03-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	PCA no dim red	Boosted Trees	5->5	lat lgn euc, manh	99.398%	4074	101	21524	54	25753	99.398%	98.69%	0.47%	99.53%	1.31%	PCA 5 Boosted Trees	-0.04%
26-03-2018	Splits = 20. Ginis diversity. Surrogate off	PCA down to 4	Tree Medium	5->4	lat lgn euc, manh	99.398%	4074	101	21524	54	25753	99.398%	98.69%	0.47%	99.53%	1.31%	PCA 4 Tree Medium	0.00%
26-03-2018		PCA down to 4	SVM Quadratic	5->4	lat lgn euc, manh	97.996%	3816	204	21421	312	25753	97.996%	92.44%	0.94%	99.06%	7.56%	PCA 4 SVM Quadratic	-1.07%
26-03-2018	Kernel scale = 0.35	PCA down to 4	SVM Fine G	5->4	lat lgn euc, manh	99.317%	4086	134	21491	42	25753	99.317%	98.98%	0.62%	99.38%	1.02%	PCA 4 SVM Fine G	0.00%
26-03-2018	Nr neighbors = 10. Cubic. Equal	PCA down to 4	KNN Cubic	5->4	lat lgn euc, manh	99.115%	4076	176	21449	52	25753	99.115%	98.74%	0.81%	99.19%	1.26%	PCA 4 KNN Cubic	-0.17%
26-03-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	PCA down to 4	Boosted Trees	5->4	lat lgn euc, manh	99.410%	4075	99	21526	53	25753	99.410%	98.72%	0.46%	99.54%	1.28%	PCA 4 Boosted Trees	-0.03%
26-03-2018	Splits = 20. Ginis diversity. Surrogate off	PCA down to 3	Tree Medium	5->3	lat lgn euc, manh	99.297%	4073	126	21499	55	25753	99.297%	98.67%	0.58%	99.42%	1.33%	PCA 3 Tree Medium	-0.10%
26-03-2018		PCA down to 3	SVM Quadratic	5->3	lat lgn euc, manh	76.038%	1197	3240	18385	2931	25753	76.038%	29.00%	14.98%	85.02%	71.00%	PCA 3 SVM Quadratic	-23.24%
26-03-2018	Kernel scale = 0.35	PCA down to 3	SVM Fine G	5->3	lat lgn euc, manh	99.247%	4086	152	21473	42	25753	99.247%	98.98%	0.70%	99.30%	1.02%	PCA 3 SVM Fine G	-0.07%
26-03-2018	Nr neighbors = 10. Cubic. Equal	PCA down to 3	KNN Cubic	5->3	lat lgn euc, manh	99.305%	4081	132	21493	47	25753	99.305%	98.86%	0.61%	99.39%	1.14%	PCA 3 KNN Cubic	0.02%
26-03-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	PCA down to 3	Boosted Trees	5->3	lat lgn euc, manh	99.309%	4070	120	21505	58	25753	99.309%	98.59%	0.55%	99.45%	1.41%	PCA 3 Boosted Trees	-0.13%
26-03-2018	Splits = 20. Ginis diversity. Surrogate off	PCA down to 2	Tree Medium	5->2	lat lgn euc, manh	99.293%	4082	136	21489	46	25753	99.293%	98.89%	0.63%	99.37%	1.11%	PCA 2 Tree Medium	-0.10%
26-03-2018		PCA down to 2	SVM Quadratic	5->2	lat lgn euc, manh	70.035%	924	4513	17112	3204	25753	70.035%	22.38%	20.87%	79.13%	77.62%	PCA 2 SVM Quadratic	-29.30%
26-03-2018	Kernel scale = 0.35	PCA down to 2	SVM Fine G	5->2	lat lgn euc, manh	99.181%	4075	158	21467	53	25753	99.181%	98.72%	0.73%	99.27%	1.28%	PCA 2 SVM Fine G	-0.14%
26-03-2018	Nr neighbors = 10. Cubic. Equal	PCA down to 2	KNN Cubic	5->2	lat lgn euc, manh	99.305%	4080	131	21494	48	25753	99.305%	98.84%	0.61%	99.39%	1.16%	PCA 2 KNN Cubic	0.02%
26-03-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	PCA down to 2	Boosted Trees	5->2	lat lgn euc, manh	99.293%	4074	128	21497	54	25753	99.293%	98.69%	0.59%	99.41%	1.31%	PCA 2 Boosted Trees	-0.14%
26-03-2018	Splits = 20. Ginis diversity. Surrogate off	no PCA	Tree Medium	5/5	lat lgn euc, manh	99.394%	4080	108	21517	48	25753	99.394%	98.84%	0.50%	99.50%	1.16%	Tree Medium	
26-03-2018		no PCA	SVM Quadratic	5/5	lat lgn euc, manh	99.056%	4053	168	21457	75	25753	99.056%	98.18%	0.78%	99.22%	1.82%	SVM Quadratic	
26-03-2018	Kernel scale = 0.35	no PCA	SVM Fine G	5/5	lat lgn euc, manh	99.320%	4085	132	21493	43	25753	99.320%	98.96%	0.61%	99.39%	1.04%	SVM Fine G	
26-03-2018	Nr neighbors = 10. Cubic. Equal	no PCA	KNN Cubic	5/5	lat lgn euc, manh	99.286%	4083	139	21486	45	25753	99.286%	98.91%	0.64%	99.36%	1.09%	KNN Cubic	
26-03-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	no PCA	Boosted Trees	5/5	lat lgn euc, manh	99.437%	4078	95	21530	50	25753	99.437%	98.79%	0.44%	99.56%	1.21%	Boosted Trees	

Cam2Lidar. Data splitted for first 5000 out of 27373 and validated with 10-fold crossvalidation																	
Setup						RESULT											
Date	Information	Comment	ML Algorithm		Features	Accuracy	TP	FP	TN	FN	Sum	Accuracy	TP	FP	TN	FN	Change
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA no dim reduc	Tree coarse	6->6	lat lgn euc, manhattan, jaccard,angle	99.080%	1014	28	3940	18	5000	99.080%	98.26%	0.71%	99.29%	1.74%	-0.18%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA no dim reduc	SVM Fine G	6->6	lat lgn euc, manhattan, jaccard,angle	98.880%	999	23	3945	33	5000	98.880%	96.80%	0.58%	99.42%	3.20%	-0.62%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA no dim reduc	KNN Cosine	6->6	lat lgn euc, manhattan, jaccard,angle	99.060%	1011	26	3942	21	5000	99.060%	97.97%	0.66%	99.34%	2.03%	0.81%
28/03/2018	30 Learners	KPCA no dim reduc	Bagged Trees	6->6	lat lgn euc, manhattan, jaccard,angle	98.980%	1011	30	3938	21	5000	98.980%	97.97%	0.76%	99.24%	2.03%	-0.56%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 5 dim	Tree coarse	6->5	lat lgn euc, manhattan, jaccard,angle	98.780%	997	26	3942	35	5000	98.780%	96.61%	0.66%	99.34%	3.39%	-0.48%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 5 dim	SVM Fine G	6->5	lat lgn euc, manhattan, jaccard,angle	98.600%	981	19	3949	51	5000	98.600%	95.06%	0.48%	99.52%	4.94%	-0.90%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 5 dim	KNN Cosine	6->5	lat lgn euc, manhattan, jaccard,angle	98.820%	994	21	3947	38	5000	98.820%	96.32%	0.53%	99.47%	3.68%	0.57%
28/03/2018	30 Learners	KPCA to 5 dim	Bagged Trees	6->5	lat lgn euc, manhattan, jaccard,angle	98.820%	1001	28	3940	31	5000	98.820%	97.00%	0.71%	99.29%	3.00%	-0.72%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 4 dim	Tree coarse	6->4	lat lgn euc, manhattan, jaccard,angle	98.800%	996	24	3944	36	5000	98.800%	96.51%	0.60%	99.40%	3.49%	-0.46%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 4 dim	SVM Fine G	6->4	lat lgn euc, manhattan, jaccard,angle	98.420%	971	18	3950	61	5000	98.420%	94.09%	0.45%	99.55%	5.91%	-1.09%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 4 dim	KNN Cosine	6->4	lat lgn euc, manhattan, jaccard,angle	98.880%	999	23	3945	33	5000	98.880%	96.80%	0.58%	99.42%	3.20%	0.63%
28/03/2018	30 Learners	KPCA to 4 dim	Bagged Trees	6->4	lat lgn euc, manhattan, jaccard,angle	98.780%	997	26	3942	35	5000	98.780%	96.61%	0.66%	99.34%	3.39%	-0.76%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 3 dim	Tree coarse	6->3	lat lgn euc, manhattan, jaccard,angle	98.760%	994	24	3944	38	5000	98.760%	96.51%	0.60%	99.40%	3.49%	-0.50%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 3 dim	SVM Fine G	6->3	lat lgn euc, manhattan, jaccard,angle	98.160%	958	18	3950	74	5000	98.160%	92.83%	0.45%	99.55%	7.17%	-1.35%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 3 dim	KNN Cosine	6->3	lat lgn euc, manhattan, jaccard,angle	98.440%	975	21	3947	57	5000	98.440%	94.48%	0.53%	99.47%	5.52%	0.18%
28/03/2018	30 Learners	KPCA to 3 dim	Bagged Trees	6->3	lat lgn euc, manhattan, jaccard,angle	98.540%	990	31	3937	42	5000	98.540%	95.93%	0.78%	99.22%	4.07%	-1.00%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 2 dim	Tree coarse	6->2	lat lgn euc, manhattan, jaccard,angle	98.680%	991	25	3943	41	5000	98.680%	96.03%	0.63%	99.37%	3.97%	-0.58%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 2 dim	SVM Fine G	6->2	lat lgn euc, manhattan, jaccard,angle	98.000%	949	17	3951	83	5000	98.000%	91.96%	0.43%	99.57%	8.04%	-1.51%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 2 dim	KNN Cosine	6->2	lat lgn euc, manhattan, jaccard,angle	98.480%	977	21	3947	55	5000	98.480%	94.67%	0.53%	99.47%	5.33%	0.22%
28/03/2018	30 Learners	KPCA to 2 dim	Bagged Trees	6->2	lat lgn euc, manhattan, jaccard,angle	98.340%	983	34	3934	49	5000	98.340%	95.25%	0.86%	99.14%	4.75%	-1.21%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	No KPCA applied	Tree coarse	6/6	lat lgn euc, manhattan, jaccard,angle	99.260%	1020	25	3943	12	5000	99.260%	98.84%	0.63%	99.37%	1.16%	
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	No KPCA applied	SVM Fine G	6/6	lat lgn euc, manhattan, jaccard,angle	99.500%	1024	17	3951	8	5000	99.500%	99.22%	0.43%	99.57%	0.78%	
28/03/2018	Nr neighbors = 10. Cosine. Equal	No KPCA applied	KNN Cosine	6/6	lat lgn euc, manhattan, jaccard,angle	98.260%	1025	80	3888	7	5000	98.260%	99.32%	2.02%	97.98%	0.68%	
28/03/2018	30 Learners	No KPCA applied	Bagged Trees	6/6	lat lgn euc, manhattan, jaccard,angle	99.540%	1024	15	3953	8	5000	99.540%	99.22%	0.38%	99.62%	0.78%	

RaCam2Lidar. Data splitted for first 5000 out of 27373 and validated with 10-fold crossvalidation

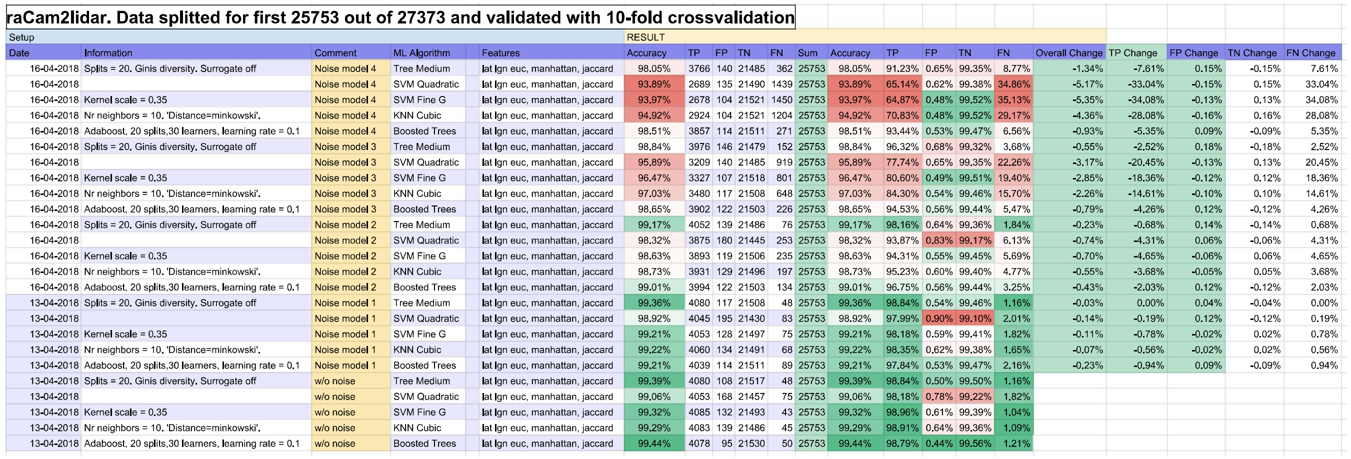
Setup						RESULT												
Date	Information	Comment	ML Algorithm		Features	Accuracy	TP	FP	TN	FN	Sum	Accuracy	TP	FP	TN	FN	Algorithm	Change
26/03/2018	Splits = 4. Ginis diversity. Surrogate off	KPCA no dim red	Tree coarse	5->5	lat lgn euc, manhattan, jaccard	99.400%	840	23	4130	7	5000	99.400%	99.17%	0.55%	99.45%	0.83%	KPCA 5 Tree Coarse	-0.16%
26/03/2018		KPCA no dim red	SVM Cubic	5->5	lat lgn euc, manhattan, jaccard	99.440%	833	14	4139	14	5000	99.440%	98.35%	0.34%	99.66%	1.65%	KPCA 5 SVM Cubic	0.42%
26/03/2018	Kernel scale = 0.35	KPCA no dim red	SVM Fine G	5->5	lat lgn euc, manhattan, jaccard	99.440%	838	19	4134	9	5000	99.440%	98.94%	0.46%	99.54%	1.06%	KPCA 5 SVM Gaussian	0.00%
26/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA no dim red	KNN Cosine	5->5	lat lgn euc, manhattan, jaccard	99.460%	839	19	4134	8	5000	99.460%	99.06%	0.46%	99.54%	0.94%	KPCA 5 KNN Cosine	2.16%
26/03/2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	KPCA no dim red	Boosted Trees	5->5	lat lgn euc, manhattan, jaccard	99.260%	832	22	4131	15	5000	99.260%	98.23%	0.53%	99.47%	1.77%	KPCA 5 Trees Boosted	-0.32%
26/03/2018	Splits = 4. Ginis diversity. Surrogate off	KPCA down to 4	Tree coarse	5->4	lat lgn euc, manhattan, jaccard	99.400%	840	23	4130	7	5000	99.400%	99.17%	0.55%	99.45%	0.83%	KPCA 4 Tree Coarse	-0.16%
26/03/2018		KPCA down to 4	SVM Cubic	5->4	lat lgn euc, manhattan, jaccard	99.400%	831	14	4139	16	5000	99.400%	98.11%	0.34%	99.66%	1.89%	KPCA 4 SVM Cubic	0.38%
26/03/2018	Kernel scale = 0.35	KPCA down to 4	SVM Fine G	5->4	lat lgn euc, manhattan, jaccard	99.480%	840	19	4134	7	5000	99.480%	99.17%	0.46%	99.54%	0.83%	KPCA 4 SVM Gaussian	0.04%
26/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA down to 4	KNN Cosine	5->4	lat lgn euc, manhattan, jaccard	99.440%	838	19	4134	9	5000	99.440%	98.94%	0.46%	99.54%	1.06%	KPCA 4 KNN Cosine	2.14%
26/03/2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	KPCA down to 4	Boosted Trees	5->4	lat lgn euc, manhattan, jaccard	99.340%	836	22	4131	11	5000	99.340%	98.70%	0.53%	99.47%	1.30%	KPCA 4 Trees Boosted	-0.24%
26/03/2018	Splits = 4. Ginis diversity. Surrogate off	KPCA down to 3	Tree coarse	5->3	lat lgn euc, manhattan, jaccard	99.420%	841	23	4130	6	5000	99.420%	99.29%	0.55%	99.45%	0.71%	KPCA 3 Tree Coarse	-0.14%
26/03/2018		KPCA down to 3	SVM Cubic	5->3	lat lgn euc, manhattan, jaccard	99.400%	835	18	4135	12	5000	99.400%	98.58%	0.43%	99.57%	1.42%	KPCA 3 SVM Cubic	0.38%
26/03/2018	Kernel scale = 0.35	KPCA down to 3	SVM Fine G	5->3	lat lgn euc, manhattan, jaccard	99.520%	842	19	4134	5	5000	99.520%	99.41%	0.46%	99.54%	0.59%	KPCA 3 SVM Gaussian	0.08%
26/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA down to 3	KNN Cosine	5->3	lat lgn euc, manhattan, jaccard	99.480%	839	18	4135	8	5000	99.480%	99.06%	0.43%	99.57%	0.94%	KPCA 3 KNN Cosine	2.18%
26/03/2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	KPCA down to 3	Boosted Trees	5->3	lat lgn euc, manhattan, jaccard	99.320%	835	22	4131	12	5000	99.320%	98.58%	0.53%	99.47%	1.42%	KPCA 3 Trees Boosted	-0.26%
26/03/2018	Splits = 4. Ginis diversity. Surrogate off	KPCA down to 2	Tree coarse	5->2	lat lgn euc, manhattan, jaccard	99.380%	838	22	4131	9	5000	99.380%	98.94%	0.53%	99.47%	1.06%	KPCA 2 Tree Coarse	-0.18%
26/03/2018		KPCA down to 2	SVM Cubic	5->2	lat lgn euc, manhattan, jaccard	99.200%	824	17	4136	23	5000	99.200%	97.28%	0.41%	99.59%	2.72%	KPCA 2 SVM Cubic	0.18%
26/03/2018	Kernel scale = 0.35	KPCA down to 2	SVM Fine G	5->2	lat lgn euc, manhattan, jaccard	99.440%	837	18	4135	10	5000	99.440%	98.82%	0.43%	99.57%	1.18%	KPCA 2 SVM Gaussian	0.00%
26/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA down to 2	KNN Cosine	5->2	lat lgn euc, manhattan, jaccard	99.440%	837	18	4135	10	5000	99.440%	98.82%	0.43%	99.57%	1.18%	KPCA 2 KNN Cosine	2.14%
26/03/2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	KPCA down to 2	Boosted Trees	5->2	lat lgn euc, manhattan, jaccard	99.280%	835	24	4129	12	5000	99.280%	98.58%	0.58%	99.42%	1.42%	KPCA 2 Trees Boosted	-0.30%
26/03/2018	Splits = 4. Ginis diversity. Surrogate off	No KPCA applied	Tree coarse	5/5	lat lgn euc, manhattan, jaccard	99.560%	835	10	4143	12	5000	99.560%	98.58%	0.24%	99.76%	1.42%	Tree Coarse	
26/03/2018		No KPCA applied	SVM Cubic	5/5	lat lgn euc, manhattan, jaccard	99.020%	821	23	4130	26	5000	99.020%	96.93%	0.55%	99.45%	3.07%	SVM Cubic	
26/03/2018	Kernel scale = 0.35	No KPCA applied	SVM Fine G	5/5	lat lgn euc, manhattan, jaccard	99.440%	843	24	4129	4	5000	99.440%	99.53%	0.58%	99.42%	0.47%	SVM Fine G	
26/03/2018	Nr neighbors = 10. Cosine. Equal	No KPCA applied	KNN Cosine	5/5	lat lgn euc, manhattan, jaccard	97.360%	828	113	4040	19	5000	97.360%	97.76%	2.72%	97.28%	2.24%	KNN Cosine	
26/03/2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	No KPCA applied	Boosted Trees	5/5	lat lgn euc, manhattan, jaccard	99.580%	838	12	4141	9	5000	99.580%	98.94%	0.29%	99.71%	1.06%	Boosted Trees	

Cam2Lidar. Data splitted for first 5000 out of 27373 and validated with 10-fold crossvalidation

Setup						RESULT											
Date	Information	Comment	ML Algorithm		Features	Accuracy	TP	FP	TN	FN	Sum	Accuracy	TP	FP	TN	FN	Change
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA no dim reduc	Tree coarse	6->6	lat lgn euc, manhattan, jaccard,angle	99.080%	1014	28	3940	18	5000	99.080%	98.26%	0.71%	99.29%	1.74%	-0.18%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA no dim reduc	SVM Fine G	6->6	lat lgn euc, manhattan, jaccard,angle	98.880%	999	23	3945	33	5000	98.880%	96.80%	0.58%	99.42%	3.20%	-0.62%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA no dim reduc	KNN Cosine	6->6	lat lgn euc, manhattan, jaccard,angle	99.060%	1011	26	3942	21	5000	99.060%	97.97%	0.66%	99.34%	2.03%	0.81%
28/03/2018	30 Learners	KPCA no dim reduc	Bagged Trees	6->6	lat lgn euc, manhattan, jaccard,angle	98.980%	1011	30	3938	21	5000	98.980%	97.97%	0.76%	99.24%	2.03%	-0.56%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 5 dim	Tree coarse	6->5	lat lgn euc, manhattan, jaccard,angle	98.780%	997	26	3942	35	5000	98.780%	96.61%	0.66%	99.34%	3.39%	-0.48%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 5 dim	SVM Fine G	6->5	lat lgn euc, manhattan, jaccard,angle	98.600%	981	19	3949	51	5000	98.600%	95.06%	0.48%	99.52%	4.94%	-0.90%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 5 dim	KNN Cosine	6->5	lat lgn euc, manhattan, jaccard,angle	98.820%	994	21	3947	38	5000	98.820%	96.32%	0.53%	99.47%	3.68%	0.57%
28/03/2018	30 Learners	KPCA to 5 dim	Bagged Trees	6->5	lat lgn euc, manhattan, jaccard,angle	98.820%	1001	28	3940	31	5000	98.820%	97.00%	0.71%	99.29%	3.00%	-0.72%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 4 dim	Tree coarse	6->4	lat lgn euc, manhattan, jaccard,angle	98.800%	996	24	3944	36	5000	98.800%	96.51%	0.60%	99.40%	3.49%	-0.46%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 4 dim	SVM Fine G	6->4	lat lgn euc, manhattan, jaccard,angle	98.420%	971	18	3950	61	5000	98.420%	94.09%	0.45%	99.55%	5.91%	-1.09%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 4 dim	KNN Cosine	6->4	lat lgn euc, manhattan, jaccard,angle	98.880%	999	23	3945	33	5000	98.880%	96.80%	0.58%	99.42%	3.20%	0.63%
28/03/2018	30 Learners	KPCA to 4 dim	Bagged Trees	6->4	lat lgn euc, manhattan, jaccard,angle	98.780%	997	26	3942	35	5000	98.780%	96.61%	0.66%	99.34%	3.39%	-0.76%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 3 dim	Tree coarse	6->3	lat lgn euc, manhattan, jaccard,angle	98.760%	994	24	3944	38	5000	98.760%	96.51%	0.60%	99.40%	3.49%	-0.50%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 3 dim	SVM Fine G	6->3	lat lgn euc, manhattan, jaccard,angle	98.160%	958	18	3950	74	5000	98.160%	92.83%	0.45%	99.55%	7.17%	-1.35%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 3 dim	KNN Cosine	6->3	lat lgn euc, manhattan, jaccard,angle	98.440%	975	21	3947	57	5000	98.440%	94.48%	0.53%	99.47%	5.52%	0.18%
28/03/2018	30 Learners	KPCA to 3 dim	Bagged Trees	6->3	lat lgn euc, manhattan, jaccard,angle	98.540%	990	31	3937	42	5000	98.540%	95.93%	0.78%	99.22%	4.07%	-1.00%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	KPCA to 2 dim	Tree coarse	6->2	lat lgn euc, manhattan, jaccard,angle	98.680%	991	25	3943	41	5000	98.680%	96.03%	0.63%	99.37%	3.97%	-0.58%
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	KPCA to 2 dim	SVM Fine G	6->2	lat lgn euc, manhattan, jaccard,angle	98.000%	949	17	3951	83	5000	98.000%	91.96%	0.43%	99.57%	8.04%	-1.51%
28/03/2018	Nr neighbors = 10. Cosine. Equal	KPCA to 2 dim	KNN Cosine	6->2	lat lgn euc, manhattan, jaccard,angle	98.480%	977	21	3947	55	5000	98.480%	94.67%	0.53%	99.47%	5.33%	0.22%
28/03/2018	30 Learners	KPCA to 2 dim	Bagged Trees	6->2	lat lgn euc, manhattan, jaccard,angle	98.340%	983	34	3934	49	5000	98.340%	95.25%	0.86%	99.14%	4.75%	-1.21%
28/03/2018	4 splits, ginis diversity index, surrogate splits off	No KPCA applied	Tree coarse	6/6	lat lgn euc, manhattan, jaccard,angle	99.260%	1020	25	3943	12	5000	99.260%	98.84%	0.63%	99.37%	1.16%	
28/03/2018	Kernel scale = 0.5. Box constraint level 1. Multiclass oneVSone	No KPCA applied	SVM Fine G	6/6	lat lgn euc, manhattan, jaccard,angle	99.500%	1024	17	3951	8	5000	99.500%	99.22%	0.43%	99.57%	0.78%	
28/03/2018	Nr neighbors = 10. Cosine. Equal	No KPCA applied	KNN Cosine	6/6	lat lgn euc, manhattan, jaccard,angle	98.260%	1025	80	3888	7	5000	98.260%	99.32%	2.02%	97.98%	0.68%	
28/03/2018	30 Learners	No KPCA applied	Bagged Trees	6/6	lat lgn euc, manhattan, jaccard,angle	99.540%	1024	15	3953	8	5000	99.540%	99.22%	0.38%	99.62%	0.78%	

C

Algorithmic stability



raCam2lidar. Data splitted for first 25753 out of 27373 and validated with 10-fold crossvalidation

Setup					RESULT															
Date	Information	Comment	ML Algorithm	Features	Accuracy	TP	FP	TN	FN	Sum	Accuracy	TP	FP	TN	FN	Overall Change	TP Change	FP Change	TN Change	FN Change
16-04-2018	Splits = 20. Ginis diversity. Surrogate off	Noise model 4	Tree Medium	lat lgn euc, manhattan, jaccard	98.05%	3766	140	21485	362	25753	98.05%	91.23%	0.65%	99.35%	8.77%	-1.34%	-7.61%	0.15%	-0.15%	7.61%
16-04-2018		Noise model 4	SVM Quadratic	lat lgn euc, manhattan, jaccard	93.89%	2689	135	21490	1439	25753	93.89%	65.14%	0.62%	99.38%	34.86%	-5.17%	-33.04%	-0.15%	0.15%	33.04%
16-04-2018	Kernel scale = 0.35	Noise model 4	SVM Fine G	lat lgn euc, manhattan, jaccard	93.97%	2678	104	21521	1450	25753	93.97%	64.87%	0.48%	99.52%	35.13%	-5.35%	-34.08%	-0.13%	0.13%	34.08%
16-04-2018	Nr neighbors = 10. 'Distance=minkowski'.	Noise model 4	KNN Cubic	lat lgn euc, manhattan, jaccard	94.92%	2924	104	21521	1204	25753	94.92%	70.83%	0.48%	99.52%	29.17%	-4.36%	-28.08%	-0.16%	0.16%	28.08%
16-04-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	Noise model 4	Boosted Trees	lat lgn euc, manhattan, jaccard	98.51%	3857	114	21511	271	25753	98.51%	93.44%	0.53%	99.47%	6.56%	-0.93%	-5.35%	0.09%	-0.09%	5.35%
16-04-2018	Splits = 20. Ginis diversity. Surrogate off	Noise model 3	Tree Medium	lat lgn euc, manhattan, jaccard	98.84%	3976	146	21479	152	25753	98.84%	96.32%	0.68%	99.32%	3.68%	-0.55%	-2.52%	0.18%	-0.18%	2.52%
16-04-2018		Noise model 3	SVM Quadratic	lat lgn euc, manhattan, jaccard	95.89%	3209	140	21485	919	25753	95.89%	77.74%	0.65%	99.35%	22.26%	-3.17%	-20.45%	-0.13%	0.13%	20.45%
16-04-2018	Kernel scale = 0.35	Noise model 3	SVM Fine G	lat lgn euc, manhattan, jaccard	96.47%	3327	107	21518	801	25753	96.47%	80.60%	0.49%	99.51%	19.40%	-2.85%	-18.36%	-0.12%	0.12%	18.36%
16-04-2018	Nr neighbors = 10. 'Distance=minkowski'.	Noise model 3	KNN Cubic	lat lgn euc, manhattan, jaccard	97.03%	3480	117	21508	648	25753	97.03%	84.30%	0.54%	99.46%	15.70%	-2.26%	-14.61%	-0.10%	0.10%	14.61%
16-04-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	Noise model 3	Boosted Trees	lat lgn euc, manhattan, jaccard	98.65%	3902	122	21503	226	25753	98.65%	94.53%	0.56%	99.44%	5.47%	-0.79%	-4.26%	0.12%	-0.12%	4.26%
16-04-2018	Splits = 20. Ginis diversity. Surrogate off	Noise model 2	Tree Medium	lat lgn euc, manhattan, jaccard	99.17%	4052	139	21486	76	25753	99.17%	98.16%	0.64%	99.36%	1.84%	-0.23%	-0.68%	0.14%	-0.14%	0.68%
16-04-2018		Noise model 2	SVM Quadratic	lat lgn euc, manhattan, jaccard	98.32%	3875	180	21445	253	25753	98.32%	93.87%	0.83%	99.17%	6.13%	-0.74%	-4.31%	0.06%	-0.06%	4.31%
16-04-2018	Kernel scale = 0.35	Noise model 2	SVM Fine G	lat lgn euc, manhattan, jaccard	98.63%	3893	119	21506	235	25753	98.63%	94.31%	0.55%	99.45%	5.69%	-0.70%	-4.65%	-0.06%	0.06%	4.65%
16-04-2018	Nr neighbors = 10. 'Distance=minkowski'.	Noise model 2	KNN Cubic	lat lgn euc, manhattan, jaccard	98.73%	3931	129	21496	197	25753	98.73%	95.23%	0.60%	99.40%	4.77%	-0.55%	-3.68%	-0.05%	0.05%	3.68%
16-04-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	Noise model 2	Boosted Trees	lat lgn euc, manhattan, jaccard	99.01%	3994	122	21503	134	25753	99.01%	96.75%	0.56%	99.44%	3.25%	-0.43%	-2.03%	0.12%	-0.12%	2.03%
13-04-2018	Splits = 20. Ginis diversity. Surrogate off	Noise model 1	Tree Medium	lat lgn euc, manhattan, jaccard	99.36%	4080	117	21508	48	25753	99.36%	98.84%	0.54%	99.46%	1.16%	-0.03%	0.00%	0.04%	-0.04%	0.00%
13-04-2018		Noise model 1	SVM Quadratic	lat lgn euc, manhattan, jaccard	98.92%	4045	195	21430	83	25753	98.92%	97.99%	0.90%	99.10%	2.01%	-0.14%	-0.19%	0.12%	-0.12%	0.19%
13-04-2018	Kernel scale = 0.35	Noise model 1	SVM Fine G	lat lgn euc, manhattan, jaccard	99.21%	4053	128	21497	75	25753	99.21%	98.18%	0.59%	99.41%	1.82%	-0.11%	-0.78%	-0.02%	0.02%	0.78%
13-04-2018	Nr neighbors = 10. 'Distance=minkowski'.	Noise model 1	KNN Cubic	lat lgn euc, manhattan, jaccard	99.22%	4060	134	21491	68	25753	99.22%	98.35%	0.62%	99.38%	1.65%	-0.07%	-0.56%	-0.02%	0.02%	0.56%
13-04-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	Noise model 1	Boosted Trees	lat lgn euc, manhattan, jaccard	99.21%	4039	114	21511	89	25753	99.21%	97.84%	0.53%	99.47%	2.16%	-0.23%	-0.94%	0.09%	-0.09%	0.94%
13-04-2018	Splits = 20. Ginis diversity. Surrogate off	w/o noise	Tree Medium	lat lgn euc, manhattan, jaccard	99.39%	4080	108	21517	48	25753	99.39%	98.84%	0.50%	99.50%	1.16%					
13-04-2018		w/o noise	SVM Quadratic	lat lgn euc, manhattan, jaccard	99.06%	4053	168	21457	75	25753	99.06%	98.18%	0.78%	99.22%	1.82%					
13-04-2018	Kernel scale = 0.35	w/o noise	SVM Fine G	lat lgn euc, manhattan, jaccard	99.32%	4085	132	21493	43	25753	99.32%	98.96%	0.61%	99.39%	1.04%					
13-04-2018	Nr neighbors = 10. 'Distance=minkowski'.	w/o noise	KNN Cubic	lat lgn euc, manhattan, jaccard	99.29%	4083	139	21486	45	25753	99.29%	98.91%	0.64%	99.36%	1.09%					
13-04-2018	Adaboost, 20 splits,30 learners, learning rate = 0.1	w/o noise	Boosted Trees	lat lgn euc, manhattan, jaccard	99.44%	4078	95	21530	50	25753	99.44%	98.79%	0.44%	99.56%	1.21%					