CHALMERS

Time series classification for identification of radar targets

Master's thesis in Engineering Physics

JOAKIM STRANDBERG

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015

MASTER'S THESIS IN ENGINEERING PHYSICS

Time series classification for identification of radar targets

JOAKIM STRANDBERG

Department of Signals and Systems
Division of Signals Processing and Biomedical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2015

Time series classification for identification of radar targets
JOAKIM STRANDBERG

Master's thesis EX052/2015
ISSN 1652-8557
Department of Signals and Systems
Division of Signals Processing and Biomedical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Chalmers Reproservice
Gothenburg, Sweden 2015

Time series classification for identification of radar targets
Master's thesis in Engineering Physics
JOAKIM STRANDBERG
Department of Signals and Systems
Division of Signals Processing and Biomedical Engineering
Chalmers University of Technology

Abstract

This thesis investigates the feasibility of distinguishing between two classes that are similar in size and velocities in an experimental radar system. The classification is performed using feature based time series classification methods on the history of measurements of a target track, in an attempt to catch the difference in movement pattern that can be seen in the two classes. Data from two separate test sets are used to evaluate the performance and usefulness of the classification. For the classification, four different classification algorithms of different complexity are evaluated, and their theory briefly described. The algorithms considered in this thesis are: neural networks, naive Bayes, logistic regression, and support vector machines. The inputs to these classifiers are statistical features extracted from the time series of measurements. Furthermore, the properties of the extracted features are studied to understand why some features are better separators than other. In the thesis it is shown that it is possible to use the time series to classify the types of targets present in the test sets fairly accurately. It is also found that even though there is some difference in performance and behaviour of the four classification algorithms over the test sets, the limiting factor in the classification is not the algorithms and their complexity, but rather the resolving power of the features that are used.

Keywords: Feature based classification, Radar, Time series

Acknowledgements

There are a few people without whom this thesis would not have been possible. First of all there is Stefan Eriksson who last autumn proposed this thesis to me and who have been my supervisor during the project, for which I am very grateful. I would also like to thank Johannes Wintenby, my second supervisor, for all his ideas and input. Then there are of course a lot of other people at SAAB who have helped me with everything from where to find paper and installing programs, to bouncing ideas, and I wish to send my gratitude to all of them as well. Last but not least, I thank Lennart Svensson, my examiner, for support and helping me stay on the right track.

Contents

1 Introduction

Central to many radar applications is the ability to identify the source of a detection. Radar echoes can come from anything from a flock of birds to a fighter jet, or stationary objects, such as buildings. So to be able to respond appropriately, the object has to be identified by some means. In especially military radars it is important with a reliable classification to be able to make a good threat assessment, and chose the right action.

Furthermore, the amount of detections from unimportant sources, such as trees, birds, and clouds, often outnumber the important detections from planes, boats, and other threats. Therefore it is also necessary to have classification to cull the amount of information sent to the operator, to allow the operator to focus on the important detections.

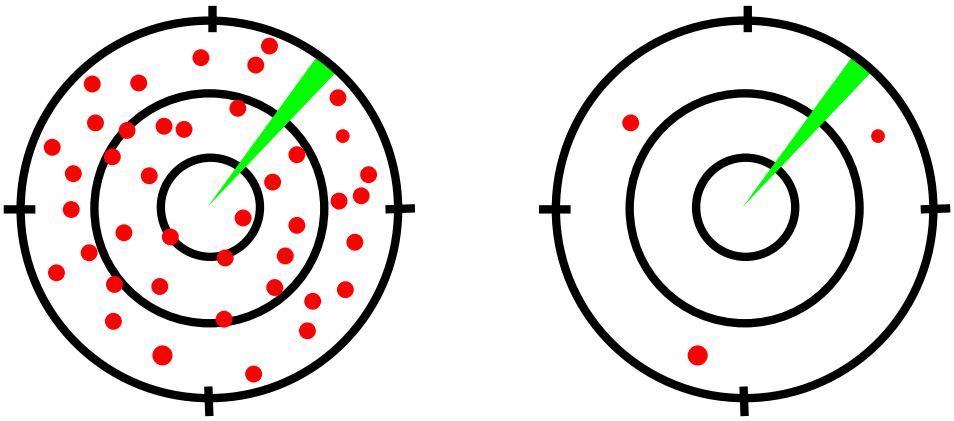


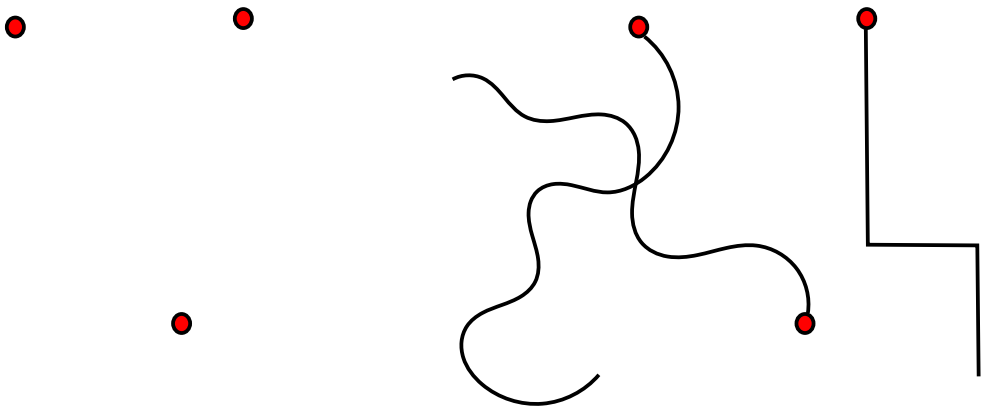
Figure 1.1: *Demonstrating why some sorting is needed. Without it, there is too much information to handle for an operator.*

In addition to the distance to an object, a radar typically measures its radial velocity and its radar cross section (RCS)[1]. These can be enough to make an educated classification about what type of object it is, and make an initial culling. For example, a flock birds fly much slower than a plane, while a missile can be distinguished from the same plane because of its size.

However, instantaneous measure of RCS and velocity is not always enough. Sometimes the physical properties of the classes are to similar to separate them, and therefore more information is needed. But the instantaneous measure is not the only information that is available. By tracking how an object and its movement over time we can also build a history of the measurements. This history and the information within can then be used to make better predictions about the class of the object.

Using this data for classification can be done in two ways. The first is that the creator of the system uses prior knowledge to define how the different classes should behave, and manually set up classification rules. For example, it could be possible to define a special manoeuvre that is typical for one of the classes, such as circulating. But this requires[2] much time and a lot of expert knowledge, and is also often trumped[2] by the second approach, machine learning.

The primary idea[3] in machine learning is that the computer should itself define the difference between the classes. Ideally, there should be as little human intervention as possible in the learning process. This avoids unwanted and unintended bias in the classification.



(a) *Without history.* (b) *With history.*

Figure 1.2: *Illustration of how the behaviour might reveal differences between two classes of objects. Without knowing the history the three dots are very similar, but knowing their paths reveal how one of them differs.*

In this thesis, the subcategory of supervised machine learning will be considered. This approach can be shortly summarized with the following steps:

1. Data on the objects is collected from sources with known class.
2. An algorithm is allowed to draw conclusions about the connection between the data and its class in order to create a function that maps input to a class, i.e. a classifier.
3. The classifier is used on new data, to make a prediction about what class it belongs to.

1.1 The problem

In the investigated problem, objects will be divided into two different classes, in the thesis denoted by class A and class B. Both of these classes are quite broad and could easily be divided into sub classes, but keeping it at two classes limits the problem to a binary classification problem, which is the most studied problem.

The classes in the problem are in single measurements indistinguishable in velocities and observed sizes. But since the objects are individually tracked, there exist time series of their past locations and velocities which will be used to classify the objects. Similarly to the illustration in Figure 1.2, it is sometimes possible to visually distinguish between the two classes, and thus the time series approach is natural. But on many parts of the tracks, the classes behave to similar for a human to differentiate them, and thus machine learning methods are chosen.

In the present thesis, only the theoretical possibilities of data driven classification is investigated. Therefore, secondary aspects such as computational load will not be considered.

1.2 On the nature of the data

The data used in this thesis comes from two separate test sets. These two sets were acquired at the same location, but at separate times therefore having different individuals of the two classes in them.

The number of tracks available is quite low: a total of 100 tracks of class A and 30 tracks of class B, divided equally between the two sets. But at the same time the tracks are long.

Therefore, to have a larger amount of samples, the tracks are split into several non-overlapping sub-tracks. This has the drawback of introducing extra interdependence between the tracks, since the same individual can give multiple samples. But keeping them non-overlapping should minimise the risk. The procedure resulted in a total of 500 samples, split 60-40 between class A and class B respectively.

2 Classification theory

This chapter will give a short general introduction to classification and to the methods chosen, and their respective theory. The last part of the chapter is dedicated to features, and why some features improve the classification results more than others.

2.1 Approaches to classification

In classification of radar targets there exist two main views: model based classification that relies on a knowledge of the dynamics of the objects, and feature based classification that instead relies on knowledge of the statistics of the objects. In this section these will be shortly described, and the choice of method motivated.

2.1.1 Model based classification

The first method of classifying radar targets is tightly connected to the task of tracking and predicting the targets position and velocities in the presence of noise and clutter detections. To only assign relevant detections to an ongoing track, the system keeps a model of how the object moves, i.e. what ranges of velocities and accelerations that are probable. Simplified, this results in a probability $P(s_{k+1} = \hat{s}|\mathbf{S}^k)$ that an observation $\hat{s}$ is the next detection of a series of detections $\mathbf{S}^k$. Only if the probability is higher than the false detection ratio will the detection be assigned to the track to avoid false tracks. Since the measurements in general contain some noise, the association process is often combined[4] with filtering the measurements to find an approximation of the true position, using for example Kalman filters.

To also couple classification to the tracking and filtering, the tracking software is given multiple filters, on for each target class. These are run in parallel [5], each computing the predicted positions according to that model. The class of the model that best describes the behaviour of the target is then assigned to the target.

To be able to create the models that are needed for the classification quite accurate knowledge of the dynamics of the classes and how they differ. Situations that is suitable[4] for model based classification are for example differing between ballistic missiles, being only affected by gravitation and drag, and rockets that can affect their speed by propulsion. Jet fighters and civilian crafts is also distinguishable, since the former have a higher maximum speed and can make sharper turns.

For the classification task in this thesis, the knowledge of the classes and their has been insufficient to create two models that were separate enough. Therefore, model based classification has not been possible.

2.1.2 Feature based classification

Feature based classification, also known as statistical classification, sees a wide usage in the data mining community, where large amounts of time series is processed. In most cases, the dynamics behind how the time series fluctuate is unknown, so it is not possible to create models for how different classes behave. Therefore another way of classifying the time series is needed.

Simply put, feature based classification takes an input vector $\mathbf{y}$ and assigns a class C to it. The vector y is a set of identifying features that are extracted from the data, and there exist

many algorithms and approaches for creating the classification function, some of which will be described later in this chapter.

Regardless of which method that is chosen to classify the object, it is important that the set of features selected as input describes and differentiates the objects well[6]. By intuition, the more difference there is between the classes in the selected features, the easier it becomes for the classifier to achieve high accuracy.

2.2 Description of classification methods

When the choice of features has been made, see Section 2.3, a function that decides the class of an object given this set has to be constructed. Since data mining and machine learning has been extensively researched during the last few decades, there exist many choices for classification algorithms. The choice of algorithm can be crucial, since the different algorithms can have very differing performance depending on the data set[6], and therefore it is interesting to test multiple algorithms on the available data to determine which is best suited to the problem. During the work on this thesis, four different classifiers of varying complexity have been chosen for investigation. These are naive Bayes, logistic regression, neural networks, and support vector machines.

2.2.1 Naive Bayes

The naive Bayes is known as a very simple classification algorithm and belongs to the class of probabilistic classification algorithms. This means that the output of the algorithm is not just a class to which it likely belongs. Instead the output is a probability distribution between the available classes. I.e. a set of probabilities $P(C_i|\mathbf{y}_n)$ which tells how likely it is that a sample $\mathbf{y}_n$ belongs to the class C_i.

The name of the classifier comes from its "naive" assumption about independence between features, given that the class is known. This is a assumption that rarely holds in reality, but still the algorithm can be competitive with more sophisticated algorithms[7], even when there is large dependencies between the variables[8].

The naive Bayes algorithm solves the maximisation problem of maximising $P(C|\mathbf{y})$, where $\mathbf{y} = (y_1, ..., y_k)$ is the set of features, with respect to the class $C \in \{C_1, ..., C_m\}$. The class that maximises this probability is the most likely class for the set of features.

By Bayes' theorem the conditional probability $P(C|\mathbf{y})$ can be rewritten as:

$$P(C|\mathbf{y}) = \frac{P(\mathbf{y}|C)P(C)}{P(\mathbf{y})}. \tag{2.1}$$

Since only the relative probabilities between the classes are of interest, and in the binary case if the ratio $R = \frac{P(C_1|\mathbf{y})}{P(C_2|\mathbf{y})}$ is larger or smaller than unity, $P(\mathbf{y})$ is eliminated. $P(C)$ is either set by some assumption, for example equal occurrence of all classes, or by measuring the relative frequencies of the classes.

Furthermore, the central assumption about independence between the variables, given the class, allows for rewriting $P(\mathbf{y}|C)$ as:

$$P(\mathbf{y}|C) = \prod_{i=1}^{k} P(y_i|C). \tag{2.2}$$

With this, the classification problem can be rewritten as the following maximisation problem:

$$\hat{C} = \arg\max_{C} P(C) \prod_{i=1}^{k} P(y_i|C), \tag{2.3}$$

where $\hat{C}$ is the estimate of the class.

Since the true conditional probabilities $P(y_i|C)$ is unavailable in most nontrivial examples, these must be estimated from the frequencies in the training data. Using the Laplace estimation of the probabilities to avoid 0 products[9], we have the approximate probabilities:

$$\tilde{P}(y_i|C) = \frac{\text{count}(y_i|C) + 1}{\text{count}(C) + m}, \tag{2.4}$$

where m is the number of classes differentiated between.

Goodness properties of the naive Bayes classifier

The naive Bayes algorithm has several positive features that make it a good choice. Firstly, it is easy to train the algorithm: given a set of training data, to calculate all conditional probabilities it is sufficient to run through the set once, which allows for fast training times. And since the product of probabilities can be expressed as sums of logarithms, the execution time is also short.

Another positive feature of the naive Bayes classifier is its ability to handle lost or missing data points. If for example the j:th feature is unavailable for an object for some reason, its class can be determined as the class that maximises the product of probabilities skipping the missing variable:

$$\hat{C} = \arg\max_C P(C) \prod_{i \neq j} \tilde{P}(y_i|C). \tag{2.5}$$

2.2.2 Continuous variables in the naive Bayes classifier

Originally, the naive Bayes has been used in the setting of of discrete variables with a limited set of values. It is for example popular in the text retrieval community[10] where the variables is mostly discrete, one such variable could be if a certain word is in the text. Then it is simple to assign a probability of the sample having a certain value of the variable for each class, by simply looping through the possible classes and values.

But since its introduction, the algorithm has been extended into the regime of continuous variables and variables with an arbitrary number of possible values. Then it is no longer feasible to assign the probabilities by simply looping through the possible values. There exists three main methods[11] for extending the naive Bayes: *the normal method*, *the kernel method*, and *the discretisation method*. These will be briefly described in this section.

The normal method

The normal method assumes that the variables are normally distributed given each class. That is, given a class C, the distribution of variable y_i can be described as:

$$P(y_i|C) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{(y_i-\mu)^2}{2\sigma^2}}, \tag{2.6}$$

where μ and σ is the mean and variance, respectively, of y_i given the class C. These are the ample mean and variance from the training data when just considering all values belonging to class C.

The method works well when the assumption of normal distribution is valid, and under those circumstances it gives good results and requires short training times[11].

The kernel method

Instead of approximating the distribution of $P(y_i|C)$ with one single normal distribution, the kernel method approximates the distribution by a sum of functions centred around the data points in the training data. These functions, or kernels, can of course also be normal distributions.

With this method, more arbitrary distributions can be approximated[12], giving the classifier the possibility of making non-linear decisions. But it also comes at a higher cost of training the classifier and calculating the probabilities.

The discretisation method

As the name implies, the discretisation method is based around discretising the continous variables. In the most simple case a single threshold is choosen for each variable, and then the probability of being above or below this threshold is found for each class and used as the basis for classification. For maximum performance, the threshold for a variable should be chosen so that it divides the initial set S into two subsets S_1 and S_2 so as to minimize the induced class entropy[13]:

$$E = \frac{|S_1|}{|S|}\text{Ent}(S_1) + \frac{|S_2|}{|S|}\text{Ent}(S_2). \tag{2.7}$$

The class entropy on a set S is just the standard entropy of the probability $P(C_i, S)$ of the classes on that set:

$$\text{Ent}(S) = -\sum_{i=1}^{m} P(C_i, S)\log(P(C_i, S)). \tag{2.8}$$

2.2.3 Logistic regression

Logistic regression is considered to be among the most simple classification algorithms, especially in the case of binary classification. Logistic regression takes a different approach to the classification problem than that of the naive Bayes; whereas the Bayesian approach is to model the probability of a measurement $\mathbf{y}$ given a class C to then use Bayes' rule to arrive at the most probable class, logistic regression approximates the probability $P(C|\mathbf{y})$ directly. These two approaches are called generative and discriminative models, respectively.

The logistic regression classifier is based around the logistic function, from which it derives its name:

$$F(\mathbf{y}) = \frac{1}{1 + e^{-(k_0 + \mathbf{k}\cdot\mathbf{y})}}. \tag{2.9}$$

This function $F(\mathbf{y})$ has values in the range $[0, 1]$ and can be considered the probability that the class C belonging to the input $\mathbf{y}$ is C_1 rather than C_0, i.e. $P(C = C_1) = F(\mathbf{y})$. So $F(\mathbf{y}) = 1$ means that the classifier is certain that the class is C_1, whereas $F(\mathbf{y}) = 0$ means that the class is C_0 according to the algorithm. Values between 0 and 1 represents more uncertain classifications.

Logistic regression is very tightly connected to linear regression, which can be seen by considering that the binary classification problem of deciding between class C_1 and C_2 with probabilities $p_1 = P(C = C_1)$ and $p_2 = P(C = C_2)$. The ratio between the probabilities for the two classes then becomes:

$$\frac{p_1}{p_2} = \frac{p_1}{1 - p_1} = e^{(k_0 + \mathbf{k}\cdot\mathbf{y})}. \tag{2.10}$$

Taking the natural logarithm of this fraction we have:

$$\ln \frac{p_1}{p_0} = k_0 + \mathbf{k} \cdot \mathbf{y}, \tag{2.11}$$

which is a simple linear regression problem.

2.2.4 Artificial neural networks

Artificial neural networks (ANN) find their inspiration in the structure of the animal brain, in which an input signal, a stimuli, is mapped to several neurons which will send out an impulse if the stimuli is strong enough. These neurons are in turn coupled to other neurons which also may fire a signal if enough of the input neurons have sent out their pulse. This network of neurons and its structure is the basis of animal intelligence; no single neuron can be considered intelligent, but the complete structure of coupled neurons can create an intelligent creature.

In the most simple case, an artificial neural network consists of a single neuron, which is a linear mapping of its k inputs $f(\mathbf{y}) = \Theta\left(\sum_{i=1}^{k} \omega_i y_i + \mu\right)$. ω_i is the input weights and Θ is an activation function, for example a sigmoid, limiting the output into the wanted regime, commonly$f(\mathbf{y}) \in [0, 1]$. For convenience, the constant term μ can be seen as a weight times an artificial neuron y_{k+1} that is always set to be 1. Then the neurons function can be written as:

$$f(\mathbf{y}) = \Theta\left(\sum_{i=1}^{k+1} \omega_i y_i\right). \tag{2.12}$$

A full ANN consists of multiple neurons, organized into N layers, such that the output of layer n is the input of layer $n+1$. In this manner the neurons of layer $n+1$ is only dependent of the outputs of layer n and not on the other neurons in the same layer. This is what is commonly called a feed forward network and can be mathematically represented as:

$$f_i^n(\mathbf{y}) = \Theta\left(\sum_j \omega_{i,j}^n s_j^{n-1}\right), \tag{2.13}$$

where

$$s_j^{n-1} = \begin{cases} f_i^{n-1}(\mathbf{y}), & n-1 > 0 \\ y_i, & n = 0 \end{cases}. \tag{2.14}$$

For ease of notation in the following section, the input to neuron i in layer n will be notated as $x_i^n = \sum_j \omega_{i,j}^n s_j^{n-1}$.

2.2.5 Training neural networks

Since the idea of a neural network is to let the structure build up itself, a method of training the network has to be introduced. The most popular method of training an ANN is called the error backpropagation algorithm which has since its proposal grown to become the prevalent method for training neural network. Many later studies are mostly concerned with the exact implementation of the algorithm rather than proposing alternatives.

The backpropagation algorithm is based on gradient descent. An energy function is defined, and then the algorithm minimises the energy function over the training set, with respect to the weights of the layers.

The energy function given input pattern p is defined as

$$E_p = \frac{1}{2}\sum_i \left(t_i^p - s_i^{N,p}\right)^2, \tag{2.15}$$

where t_i^p the desired, or true, output given pattern p and $s_i^{N,p}$ the actual output from the neuron i in layer N, the output layer.

By applying the chain rule we have that

$$\frac{\partial E}{\partial \omega_{i,j}^N} = \frac{\partial E}{\partial s_i^N}\frac{\partial s_i^N}{\partial \omega_{i,j}^N}. \tag{2.16}$$

Now we have that

$$\frac{\partial s_i^N}{\partial \omega_{i,j}^N} = \frac{\partial s_i^N}{\partial x_i^N}\frac{\partial x_i^N}{\partial \omega_{i,j}^N} = \Theta'(x_i^N)s_j^{N-1}. \tag{2.17}$$

For the output layer the other derivative is simply

$$\frac{\partial E}{\partial s_i^N} = -(t_i - s_i^N). \tag{2.18}$$

For layers other than the output layer $\frac{\partial s_i^n}{\partial \omega_{i,j}^n}$ retains the same form, but $\frac{\partial E}{\partial s_i^n}$ becomes more complicated, as it is dependent on following layers:

$$\frac{\partial E}{\partial s_i^n} = \sum_j \frac{\partial E}{\partial s_j^{n+1}}\frac{\partial s_j^{n+1}}{\partial s_i^n} = \sum_j \frac{\partial E}{\partial s_j^{n+1}}\frac{\partial s_j^{n+1}}{\partial x_j^{n+1}}\frac{\partial x_j^{n+1}}{\partial s_i^n} = \sum_j \frac{\partial E}{\partial s_j^{n+1}}\Theta'(x_j^{n+1})\omega_{j,i}. \tag{2.19}$$

Equation (2.19) is the reason the algorithm is called the backpropagation algorithm: to compute the derivatives, and thus the weight corrections, of layer n it is necessary to know the derivatives of layer $n+1$. Therefore one has to start at the output layer, and then successively work toward the input layer.

Once all derivatives are worked out the weights are updated according to:

$$\Delta\omega_{i,j} = -\delta\frac{\partial E}{\partial \omega_{i,j}}, \tag{2.20}$$

where $\delta > 0$.

To make the training converge faster several variants and ad hoc methods have been proposed[14]. Among these are variable step length δ, or having a momentum term in the weight update equation (2.20). But regardless of these changes the main behaviour of the algorithm still stays the same.

Stopping criterion

When training a neural network it is important to know when to stop. Neural networks have many parameters and if trained for too long the algorithm begin to overfit the training set, achieving very good performance on the training data, but poor performance on anything else.

To avoid this a stopping criterion has to be implemented. This is done by using a test set. After each update of the weights in the network, its performance is tried on the test set. Then when the test set performance increases too many consecutive iterations the algorithm is stopped. After stopping the training algorithm, the set of weights that gave the lowest error on the test set is chosen as the trained neural net.

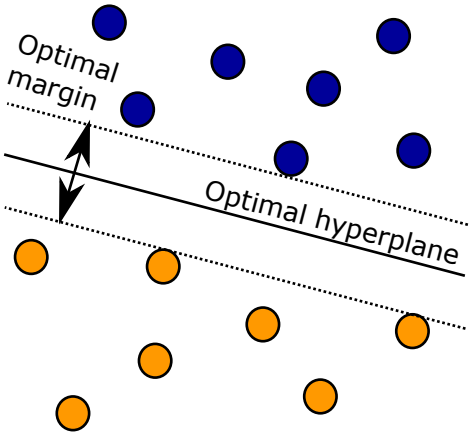


Figure 2.1: *Illustration of the optimal separating hyper plane and its corresponding margin in the totally linearly separable case.*

2.2.6 Support vector machines

Support vector machines (SVM) belongs to the group of algorithm called kernel methods. In most other classification algorithms, some model parameters are fitted to the training data after which the training set has played its role and is discarded. Only the parameters are saved for future classifications. In kernel algorithms, on the other hand, the training data is saved and used to make predictions about new data.

Kernel methods are based around kernel functions, which are functions of the form:

$$k(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{y}). \tag{2.21}$$

These are inner products in a feature space to which the inputs are transformed via the mapping $\phi(\mathbf{x})$. This feature space must not necessarily be of the same dimensionality as the input space, and often have more dimensions.

As an example of how a kernel representation can be formed we can take the function $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^{\mathrm{T}}\mathbf{y})^2$ in the two dimensional input space $\mathbf{x} = (x_1, x_2)^{\mathrm{T}}$. Expanding this we have

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= (x_1 y_1 + x_2 y_2)^2 \\ &= x_1^2 y_1^2 + 2 x_1 y_1 x_2 y_2 + x_2^2 y_2^2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)(y_1^2, \sqrt{2} y_1 y_2, y_2^2)^{\mathrm{T}}. \end{aligned} \tag{2.22}$$

Thus, the function can be written on kernel form with the mapping $\phi(\mathbf{x}) = (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^{\mathrm{T}}$.

Essentially, the support vector algorithm finds the optimal hyperplane in the chosen feature space that separate the classes with the widest margin. The margin is the closest distance from the separating plane to any of the samples, see Figure 2.1 for an illustration.

The separating hyperplane will be on the form

$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \phi(\mathbf{x}) + b = 0. \tag{2.23}$$

Note that the distance between a point $\mathbf{x}$ and the hyperplane will be

$$d = \frac{f(\mathbf{x})}{|\mathbf{w}|}. \tag{2.24}$$

So finding the optimal hyperplane is equivalent to maximising the smallest d.

There is one degree of freedom in (2.24) since the distance is unaffected by a rescaling of both $\mathbf{w}$ and b by the same constant factor. This can be used to set $|f(x)| = 1$ for the points that are closest to the surface. Thus, finding the optimal hyperplane will be the optimisation problem of finding the $\mathbf{w}$ that minimises $|\mathbf{w}|^2$ under the constraint

$$t_i f(x_i) \geq 1, \qquad i = 1, ..., N, \tag{2.25}$$

for all N training samples. t_i is the target values, which here are ± 1, and is included to guarantee that the two classes are separated.

On this form it is evident that we have a quadratic optimisation problem with N constraints. Introducing Lagrange multipliers λ_i we can write the Lagrangian:

$$L = \frac{1}{2}|\mathbf{w}|^2 - \sum_{i=1}^{N} t_i \lambda_i \left(f(\mathbf{x}_i) - 1 \right). \tag{2.26}$$

Now, finding the stationary point of L with respect to $\mathbf{w}$,

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{N} t_i \lambda_i \phi(\mathbf{x}) = 0 \quad \rightarrow \quad \mathbf{w} = \sum_{i=1}^{N} t_i \lambda_i \phi(\mathbf{x}_i), \tag{2.27}$$

Substituting $\mathbf{w}$ from (2.27) back into $f(\mathbf{x})$ we have

$$f(\mathbf{x}) = \sum_{i=1}^{N} t_i \lambda_i \phi(\mathbf{x}_i) \phi^{\mathrm{T}}(\mathbf{x}) + b = \sum_{i=1}^{N} t_i \lambda_i k(\mathbf{x_i}, \mathbf{x}) + b, \tag{2.28}$$

which shows how the kernel methods are applied in a SVM. Also worth noticing is that while the resulting classifier is linear in the feature space, it can be nonlinear in the input space because of the mapping $\phi(\mathbf{x})$.

This structure of the SVM makes it relatively easy to train[15], but taking an inner product between a sample that should be classified and all samples in the training set could be computationally demanding if the training set is too large. However, it can be shown[16] that the following property hold for the optimisation:

$$\lambda_i \left(t_i f(\mathbf{x}_i) - 1 \right) = 0. \tag{2.29}$$

Thus, either $\lambda_i = 0$ or $t_i f(\mathbf{x}_i) = 1$. From previous assumptions, the latter condition is true only for the training samples that are closest to the hyperplane. All other samples will have $\lambda_i = 0$, and will therefore not matter in Equation (2.28). The samples that have nonzero λ are called support vectors, and are the only ones that are important in the resulting classifier. So instead of keeping all training samples after training the classifier, only the support vectors are kept.

The kernel

The chosen kernel, and the accompanying mapping, are key to how the SVM functions. For example, if the kernel is chosen to be $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^{\mathrm{T}}\mathbf{y}$, the linear kernel, the resulting classifier will be completely linear. But if a more complex kernel is chosen, the classifier can be highly nonlinear.

Since the type of kernel, and its parameters, is set manually, the SVM algorithm needs to be adjusted for every data set. There is no universal kernel working for all problems, and some initial knowledge has to be applied in the choice. However, if it is deemed that a linear kernel is insufficient, another default kernel is the radial basis function[17]. This is defined as:

$$k(\mathbf{x}, \mathbf{y}) = e^{-\gamma |\mathbf{x} - \mathbf{y}|^2}. \tag{2.30}$$

Soft margins

In general, the training data is not completely separable, and so the constraint in (2.25) cannot be fulfilled. Therefore, the SVM algorithm has to be extended, adding soft margins. In the non-separable case, the constraint is changed to:

$$\begin{aligned} t_i f(\mathbf{x}_i) &\geq 1 - \xi_i, \\ \xi_i &\geq 0. \end{aligned} \tag{2.31}$$

ξ is introduced to allow a sample to be on the wrong side of the decision surface, "softening" the margins.

The optimisation problem then becomes[18] the minimisation of:

$$\frac{1}{2}|\mathbf{w}|^2 + CF\left(\sum_{i=1}^{N} \xi_i^\sigma\right), \tag{2.32}$$

under the constraints (2.31), where C is a constant and F a monotonic convex function. This allows for the samples to be inside the margins, $0 < \xi \leq 1$, or even on the wrong side of the hyperplane, $\xi > 1$, but will penalize those samples that have nonzero ξ. The new optimisation problem is computationally hard, but will find a hyperplane that minimises[18] the deviations ξ.

2.3 Choosing good features

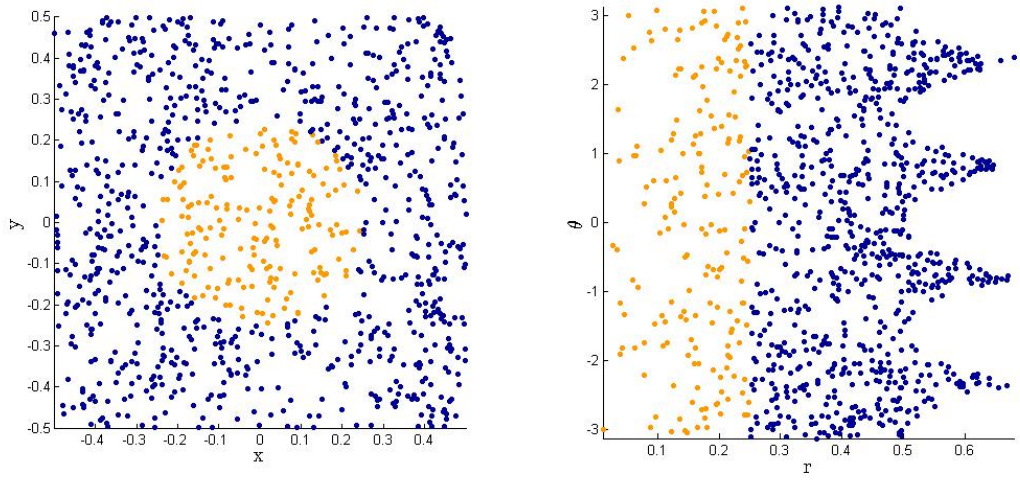
An important step in creating a well performing classifier is selecting the input features. A simple way would be to choose to input the raw measurement data, for example the whole series of positions and velocities. Intuitively this maximises the amount of information sent to the classifier, but it also has some drawbacks.

Firstly, such an input set would either limit the classifier to data of a certain length, since the classifier functions needs a predefined amount of inputs. This would either both delay an initial classification until the inputs are filled up, and also discard any information that are too old. Or it would require training and using several different classifiers, using different input lengths, to be able to consider all data.

Secondly, such a classifier easily focuses on features that are just seemingly important but is rather an artefact of the training data. For example, in the present case, the exact location of the objects relative to the radar is not of importance for its class, but in the training data there is a strong correspondence with location and class for practical reasons. In such a situation features of a higher abstraction level is more suitable, such as fluctuations in its position, and variances in its speed. A complex enough classifier, for example a multilayer neural network with many neurons, can still find these important features[19]. But the more complex a classifier is, the more training data it needs to really discern the complicated behaviours and to avoid overfitting.

Therefore, if some knowledge of the structure of the data exist, this knowledge can be applied to simplify the classification problem. As an illustration Figure 2.2 shows how a classification problem can become much easier if pretreated correctly. In the classification problem at hand, the pretreatment will be performed on time series rather than on single measurements, but the principle still stands.

Instead of using the raw measurements as inputs, statistical features will be extracted from the time series to be input to the classifiers. This will counter both the variable length of the time series, and if constructed right take care of the generalisability. A goodness property this has over using the time series itself is that it reduces the influence of noise in the measurements on the outcome of the classification[20].



(a) *Cartesian coordinates.* (b) *Polar coordinates.*

Figure 2.2: *Illustration how pretreating a problem may make it much easier if something is known about the structure of the data. In cartesian coordinates the problem is highly non-linear, but in polar coordinates a simple line divides the classes.*

2.3.1 Comparing importance of features

To sort the input variables on their importance for the classification, two approaches will be considered here. First is the wrapper methods that uses a classifier and scores the variables on their performance in the classifier. Then there is the filter methods that only considers general properties of the variables and their interdependence, without putting them into a classifier.

The first measure of interest is the linear prediction power. For the i:th variable, this is is defined as R_i^2 where R_i is the numerical estimation of the Pearson correlation coefficient for a sample[21]:

$$R_i = \frac{\sum_{k=1}^{m} (x_{k,i} - \bar{x}_i)(y_k - \bar{y})}{\sum_{k=1}^{m} (x_{k,i} - \bar{x}_i)^2 \sum_{k=1}^{m} (y_k - \bar{y})^2}. \tag{2.33}$$

This measure will be high if there is a large correlation, or anti-correlation, between the class of the samples and the value of the i:th variable, where the class is represented by 0 or 1 for this calculation. But as earlier stated, the measure does not take into account effects of the classifier, nor does it consider symbiosis with other features, which could in principle take two "useless" parameters and still extract relevant information when they are combined[21].

The second measure is designed to measure how important the variables are in the classifier. In the measure, different combinations of the variables are tried together in a classifier algorithm. The intent is that variables that are useless on their own, but rich on information in pairs or groups, will still receive a good score. The score is calculated as follows:

1. All unique combinations of n features out of the k available are formed.

2. For each feature combination a logistic regression classifier is trained and tested on the data set using 10-fold cross validation. Only the n selected features is used as input to the classifier.

3. The $\binom{k}{n}$ different sets of n features are sorted by their performance, i.e. prediction accuracy.

4. A score is calculated for each feature by summing the position of all sets that include the feature. So if feature i occurs in the 1st, 4th, and 10th sets its total score will be $S_i = 15$.

Features that consistently increases the performance of the classification will with this procedure receive a lower score than features that do not contribute as much.

3 Feature design and performance

An important aspect of the project has been to examine the features themselves. This gives the possibility to develop a deeper understanding of the dynamics of the two classes which will be a good base for further development. Therefore this chapter will present how this analysis has been performed.

First the features that have been chosen are introduced. Then the process of examining them is described leading up to their different usability.

3.1 Radar measurements

To understand the features chosen this section will describe what raw output is available from the radar. The radar measures both direction and distance to targets, which means that it can decide the position in space, $\mathbf{r} = (r, \theta, \phi)$.

The radar can also measure velocities. By looking at the slight Doppler shift induced by the radar signal being reflecting on a moving surface, a radar can measure[1] the speed that the target is heading away from the unit, v_r.

Finally, by measuring the amplitude of the reflected signal, it is possible to estimate the radar cross section (RCS), σ, of the target. Since the amplitude does not only depend on the size of the target, but also on properties on the medium, that can fluctuate, this will not be an exact measurement, but rather an educated estimate.

These measurements are the raw, and noisy, data that will be available from a single radar echo. But on that, tracking and estimation are applied, giving access to a few more quantities, and also reducing the effects of measurement noise. The estimation is performed with Kalman filters; the exact implementation is not important, and it suffice to say that after the estimation the following quantities are available:

- Estimated position, now represented in a Cartesian coordinate system for convenience: $\tilde{\mathbf{x}} = (\tilde{x}, \tilde{y}, \tilde{z})$.
- Estimated velocities, $\tilde{\mathbf{v}} = (\tilde{v}_x, \tilde{v}_y, \tilde{v}_z)$.
- Estimated RCS, $\tilde{\sigma}$.
- Estimation of the lateral acceleration, $\tilde{a}_n$, normal to the current velocity and in the xy-plane.

Since it is only these estimated quantities that will be used, the tilde will be dropped from here.

3.2 Segmenting heading and height

A concept that is used for a few of the features is segmentation, mostly of the height z and the heading θ. θ is calculated using the velocity components in the plane, v_x and v_y.

The details of how the segmentation is performed are available in Appendix A, and it will give a series of linear segments approximating the height and the heading, giving their trend. These will be denoted z_{trend} and θ_{trend}.

3.2.1 Curves and straight flight

It has been of interest to look at the behaviour during curves and straight flight separately, as is seen in Appendix *A*. A curve is defined as a stretch where the segmentation of the heading has a higher incline than a set threshold, $\frac{\mathrm{d}}{\mathrm{d}t}\theta_{\text{trend}} > \omega$.

3.3 Tested features

This section will briefly describe what features that have been considered. Each feature will also be associated with a number, which they will be identified with in figures in this chapter.

The features chosen are all statistical features that will ideally converge to some value given a track long enough. The features are also chosen to reflect different aspects of the motion of the targets, and especially the differences that by visual inspection seem to exist.

For notational purposes in this section, the variables will be considered their whole time series in a single sample. For example $\theta = (\theta(t_1), \theta(t_1), ..., \theta(t_n))$.

1. **Variance of detrended heading.** This feature is the variance of the heading when the local trend has been removed.

   $$\mathrm{Var}\left(\theta - \theta_{\text{trend}}\right)$$

   The local trend, θ_{trend}, is found through partitioning the heading data according to Appendix A. This attempts to catch the ability of objects to stay true to their course.

2. **Variance of detrended height.** This feature measure the variance of the height when, as in the previous measure, the local trend has been removed.

   $$\mathrm{Var}\left(z - z_{\text{trend}}\right)$$

3. **Variance of magnitude of lateral acceleration.**

   $$\mathrm{Var}(a_n)$$

4. **Low frequency component of RCS.** $\frac{E_{low}}{E_{tot}}$, where E_{low} is the energy in frequencies lower than 0.05 Hz, acquired from an FFT of the radar cross section.

5. **Low frequency component of heading.** Defined exactly as feature 4, but using FFT of the heading instead.

6. **Mean of magnitude of lateral acceleration during curves.**

   $$\mathrm{Mean}(a_{n,\text{curve}})$$

7. **Variance of magnitude of lateral acceleration during curves.**

   $$\mathrm{Var}(a_{n,\text{curve}})$$

8. **Mean of magnitude of lateral acceleration during straight flight.**

   $$\mathrm{Mean}(a_{n,\text{straight}})$$

9. **Variance of magnitude of lateral acceleration during straight flight.**

   $$\mathrm{Var}(a_{n,\text{straight}})$$

10. **Variance of speed.**

$$\mathrm{Var}(|\mathbf{v}|)$$

11. **Autocorrelation of the detrended heading.** The value of the autocorrelation function for the detrended heading, $\theta_{\mathrm{detrend}} = \theta - \theta_{\mathrm{trend}}$, with the lag $\tau = 5\,\mathrm{s}$.

$$\frac{\mathrm{Mean}\left[(\theta_{\mathrm{detrend}}(t) - \mu_\theta)(\theta_{\mathrm{detrend}}(t + 5\,\mathrm{s}) - \mu_\theta)\right]}{\mathrm{Var}(\theta)^2}$$

μ_θ is the mean of the detrended heading.

This measure is high if having a high detrended heading at time t corresponds well to having a high heading at time $t + 5\,\mathrm{s}$.

12. **Autocorrelation of the speed.** The value of the autocorrelation function for the speed, $v = |\mathbf{v}|$, with the lag $\tau = 5\,\mathrm{s}$.

$$\frac{\mathrm{Mean}\left[(v(t) - \mu_v)(v(t + 5\,\mathrm{s}) - \mu_v)\right]}{\mathrm{Var}(v)^2}$$

13. **Autocorrelation of the lateral acceleration.** The value of the autocorrelation function for the lateral acceleration with the lag $\tau = 5\,\mathrm{s}$.

$$\frac{\mathrm{Mean}\left[(a_n(t) - \mu_{a_n})(a_n(t + 5\,\mathrm{s}) - \mu_{a_n})\right]}{\mathrm{Var}(a_n)^2}$$

3.4 Finding the importance of the features

Initially the time series themselves were studied to gain a first oversight of the differences. This however is a time consuming and little rewarding path that is prone to mistakes and false impressions.

Therefore an alternative path was chosen: several features were constructed and then extracted from the time series. Some of these where chosen because of the initial visual examination, some from intuition about the behaviour of the classes, and some by more random selection. Then, all the tracks were divided into shorter sections for which each of the features were calculated to find their statistical distribution.

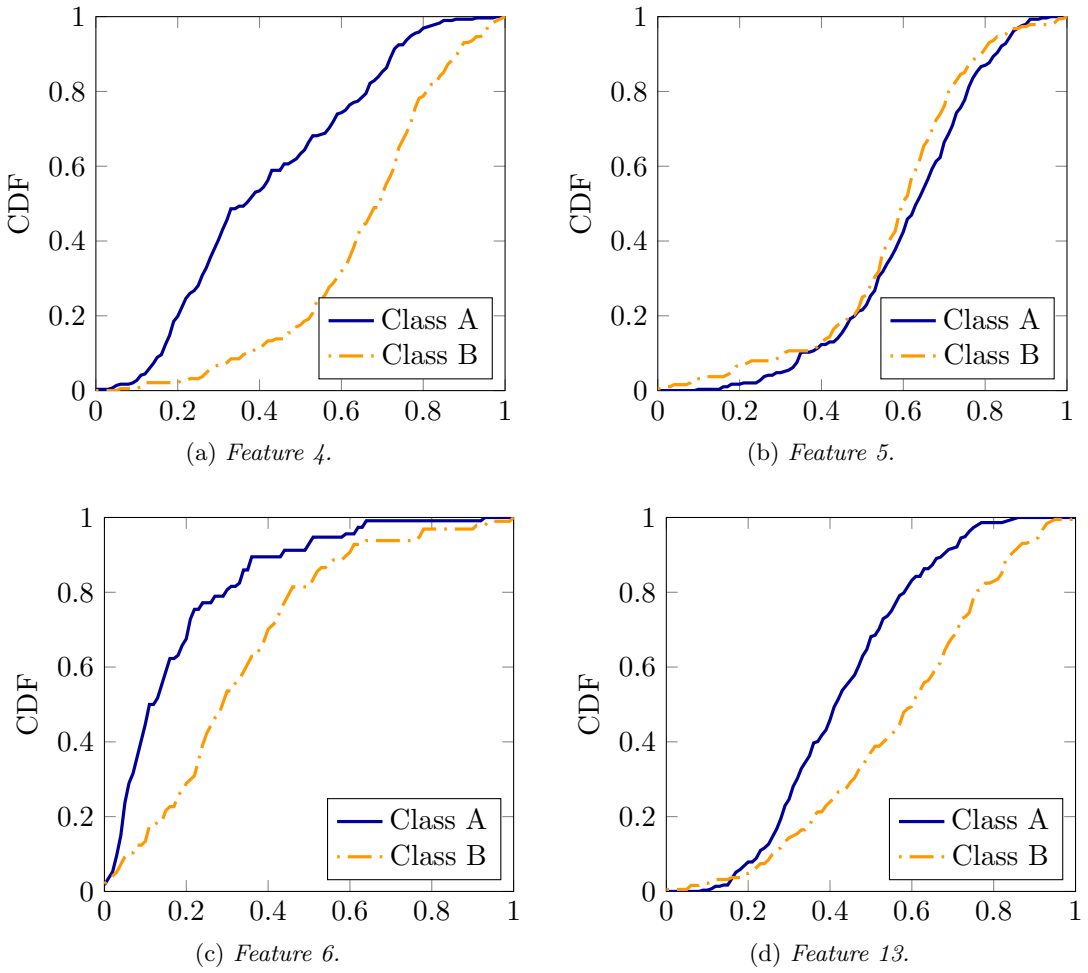
Four of the feature distributions are seen in Figure 3.1 in the form of cumulative distribution functions (CDF). Just from these CDFs it is beginning to be clear that the different features have different degrees of usability in differing between the objects: the more the CDFs for the different classes are separated, the more easily are they distinguished. Thus, for example, feature 5 in Figure 3.1b is much less useful than feature 4 of Figure 3.1a.

In figure 3.2 the linear predictive power described in Section 2.3.1 is presented for all the features. This measure is closely connected to the separability of the CDFs, and so it is not surprising to see that feature 4 outperforms the other three features shown in Figure 3.1 given that its separation is the largest.

So far we have only considered linear separability of the data: neither non-linear relationships, nor connections between the variables have been considered. Therefore the second measure described in Section 2.3.1 aims at revealing such relationships between the features and the class. As wrapper classifier for the calculation of the performance score, a neural network with a single hidden layer with 5 neurons has been used. The choice of classifier will be further discussed in the following chapter, but of the algorithms considered in this thesis, ANN is the one that has most capacity to find coupled relationships.

The resulting performance score is seen in Figure 3.3, and from how it is defined a low score hints that the feature is more beneficent for the classification. Again, feature 4 stands out as the best feature with 13 coming second. And no other feature has seen a significant difference in performance compared to Figure 3.2, pointing at that the chosen features are mainly linearly separable.

An example of the linearity of the data can be seen in Figure 3.4 in which the samples are plotted in the plane spanned by the two best features of Figures 3.2 and 3.3. In the feature plane formed by these, the two classes are to a quite high degree linearly separable, forming two clusters. But it is also evident that these two features are not enough in themselves since there is a large overlap. To resolve the class of all samples additional information is needed.



(a) *Feature 4.* (b) *Feature 5.*

(c) *Feature 6.* (d) *Feature 13.*

Figure 3.1: *Cumulative distribution functions (CDF) of features 4, 5, 6, and 13 for the two classes. The scales of the feature values are normalised with the maximum value of the respective feature over both classes.*

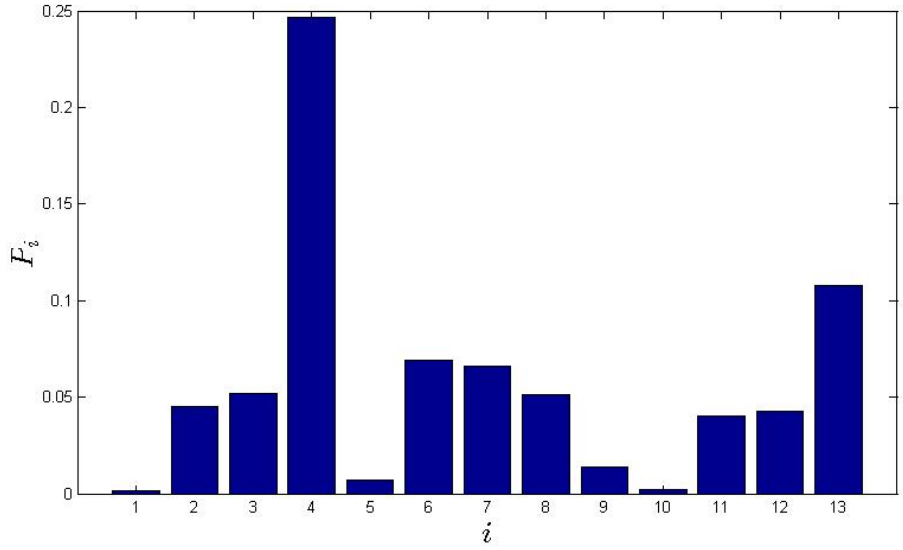


Figure 3.2: *Linear predictive power for the tested features. The higher the predictive power, the more correlation between the value of the feature and the class of the target.*

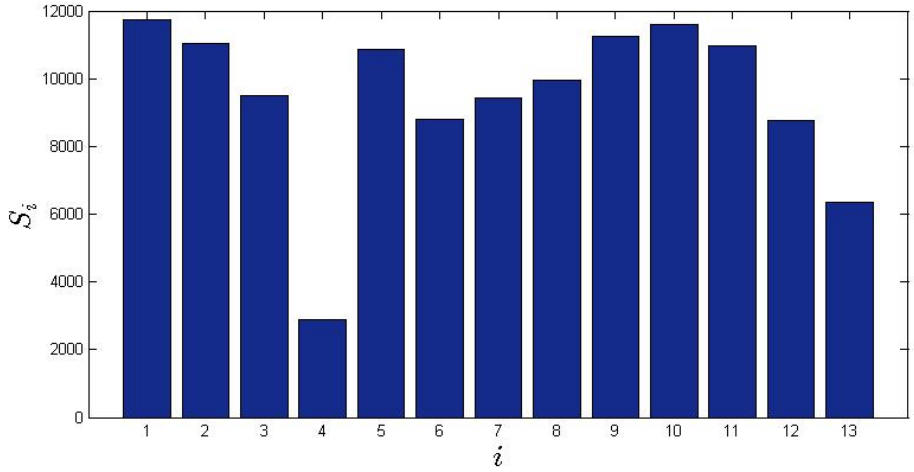


Figure 3.3: *Performance scores of the features as defined in Section 2.3.1, using groups of size $n = 3$ and a neural network classifier as wrapper. A low score means a good performance.*

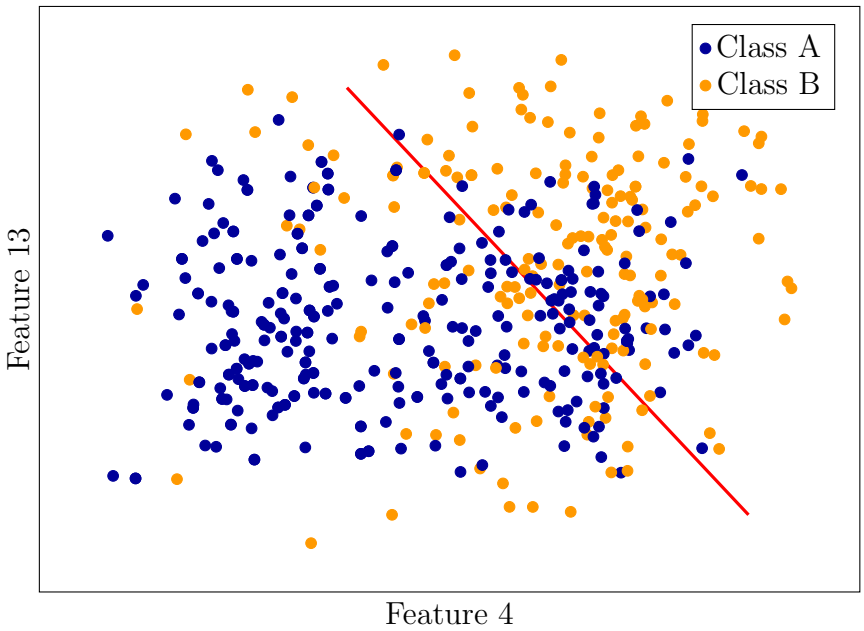


Figure 3.4: *Scatter plot of feature 4 and 13 of all samples used for training and testing. Used together they can to a good approximation separate the samples linearly. The red line shows the best separation found by logistic regression, giving a correct classification of* 73.2 %.

4 Classifier performance

In this chapter the performance of the actual classifiers will be presented and discussed. All four classification algorithms mentioned in Chapter 2 will be considered and compared.

4.1 Comparing accuracy on small data sets

Ideally, when comparing classification methods a number of large similar data sets should be available. Each of these sets is then divided into a training set and a validation set and the different classification algorithms are trained and tested on each of these pairs of sets. Comparing the results should then be enough to state which classification method that is best suited for the problem at hand.

When the amount of available data is small it can be troublesome to divide data into completely separate test sets large enough to actually train the classifier accurately. Then it might be necessary to use the same data in each test, only changing the division of data points into training and validation set. In doing so, the tests will not be independent, and exactly how this division is done and changed between the tests can affect the result of the comparison.

One of the more powerful[22][23] test methods is called the *k-fold cross-validated paired t test*. In this test the dataset is divided into k equally sized sets. Then k trials are performed using the classification algorithms. In trial i, all but set i are used as training data for each of the algorithms, and the validation is performed on set i giving a precision $p^{(i)}_{\text{algorithm}}$. To then determine if the algorithms give a significant difference in performance, the Student's t-test is applied pairwise on the distributions of precisions for the algorithms.

4.2 Comparison of the classification methods

Four different methods have been tested on the available data: Naive Bayes[1], support vector machines[2], neural networks[3], and logistic regression. Because of the small data sets available a cross-validation has been performed as described in Section 4.1, using $k = 10$ partitions.

Figure 4.1 shows the accumulated correctly and incorrectly classified instances for each of the two examined classes. The mayor difference between the classes is in the accuracy over the different classes, but total accuracy are consistent. The lower accuracy of SVM on class A is compensated for with a better performance on class B, and so on. No significant, ($p < 0.05$), difference in their total performance, i.e. accuracy over the whole set, could be found in comparing their precisions on the different partitions.

The difference in accuracy on the different classes seen in Figure 4.1 can be described by that the algorithms chooses different positions on the *Receiver Operating Characteristics* (ROC) curve. An ROC curve describes the trade-off an algorithm do between accuracy on the two classes. That the algorithms indeed would perform very similar if the same trade off had been chosen can be seen from the ROC curve in Figure 4.2. The curves for the different classifiers follow each other closely, showing that the four classifiers have very similar behaviour on this data set.

[1] Using the kernel method as described in Section 2.2.2

[2] Using radial basis function kernels.

[3] Two hidden layers: first one with five neurons, second one with three. The choice did not affect the classification significantly.

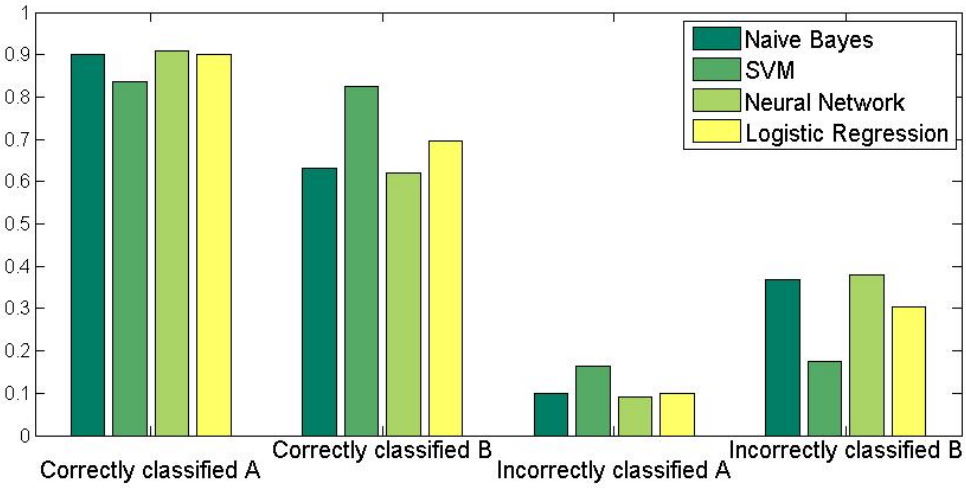


Figure 4.1: *Average accuracy of the four different classifiers that have been considered when trained and tested on the first of the two data sets, using 10-fold cross-validation. No considerable difference could be measured in the total accuracy between the algorithms.*

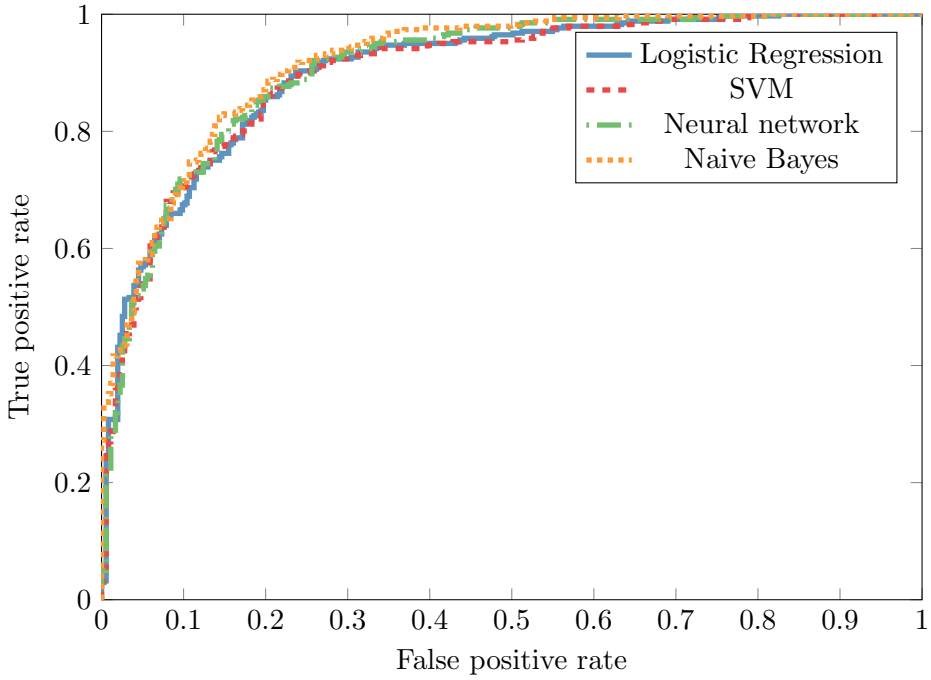


Figure 4.2: *Receiver operating characteristics curves for the four different classifiers discussed in this chapter, where class B of Figure 4.1 is considered the positive class. The curve was calculated using both available data sets.*

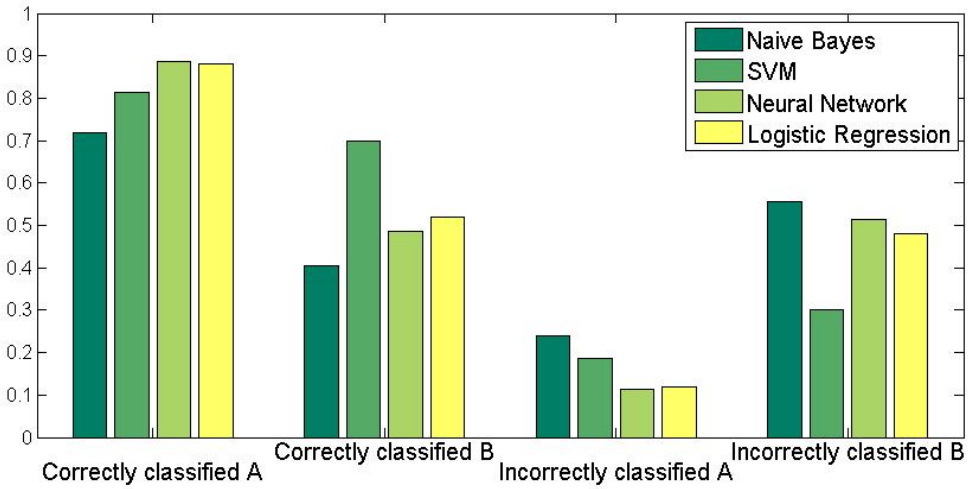


Figure 4.3: *Average accuracy of the four different classifiers when trained and tested on two completely disjunct set respectively.*

4.3 Taking classifiers to new data

A major concern when using machine learning methods has always been the risk of overfitting. As earlier mentioned this is mitigated by using 10-fold cross-validation, but because of factors in the data collection procedure there is risk that the samples in themselves have a correlation.

To see the effects of overfitting and extending to new regimes, the data from the two different trial periods were used independently. To give a comparable result to Figure 4.1 in which randomness in the algorithms were averaged out, the classifiers were tested using a modified variant of 10-fold cross-validation: they were trained on 90 % of the data from the first occasion and then validated on 10 % of the data from the second trial. Then the what parts were used were rotated as in the ordinary cross-validation.

The result of this modified cross validation can be seen in Figure 4.3. The performance result of this test differs somewhat between the algorithms, especially for the naive Bayes performs the worst in expanding to the new data set, but all algorithms have a significant decrease in accuracy. The average accuracy over both classes for both tests can be seen in Table 4.1. Also included is the average accuracy when both data sets are combined into one, on which the algorithms are tested with 10-fold cross validation. This last test have a lower accuracy than when only using the first data set, but higher than when trying to expand the classifiers outside their original regime. This is what can be expected; to gain generalisability some accuracy has to be sacrificed.

To reduce the overfitting it is sometime beneficent to remove "unnecessary" input parameters from the classifier. Features with low, or no, information mostly only add some noise that further reinforces the overfitting. To see this effect, Figure 4.4 show the accuracy of the logistic regression classifier when successively removing the worst features and retraining the classifier. The worst features are the features having the lowest predictive power in Figure 3.3.

It is clear that while removing features decreases the performance on the training set itself, it also increases the performance on the separate data set at first, showing that the classifiers are prone to over fitting. But removing too many features naturally decreases the overall performance since too much information is lost.

Table 4.1: Average total accuracy percentage over both classes of the algorithms on the different test scenarios. The first two rows show the accuracy after training and classifying on the same set. Row three and four shows the accuracy after training on one set, and then classifying on the other set. The last row shows the accuracy when the two trials are added into one large set on which 10-fold cross validation is performed.

	Classifier accuracies (%)			
Data set	NB	SVM	NN	LR
Set 1	82.8 ± 0.6	84.4 ± 0.7	83.6 ± 0.6	85.8 ± 0.6
Set 2	76.8 ± 0.9	75.9 ± 0.7	71.6 ± 1.8	78.4 ± 0.7
$1 \rightarrow 2$	58.6 ± 0.2	70.3 ± 0.2	62.8 ± 0.3	67.4 ± 0.2
$2 \rightarrow 1$	68.3 ± 0.2	73.4 ± 0.7	79.5 ± 0.2	77.8 ± 0.1
Combined	80.3 ± 0.9	80.7 ± 0.8	80.7 ± 0.5	80.3 ± 0.7

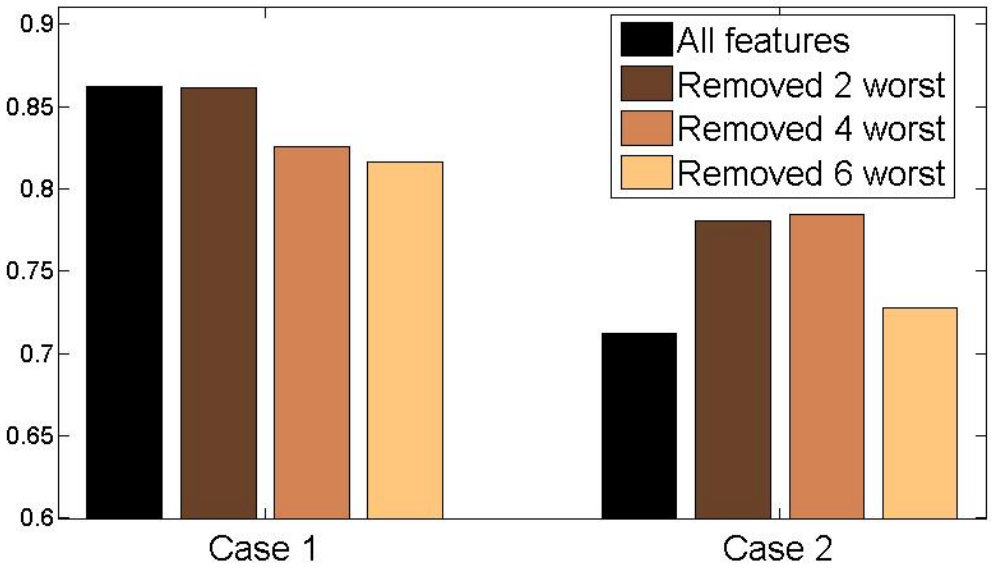


Figure 4.4: *Total accuracy of the logistic regression classifier that is 1) trained and tested on data from the same trial occasion, using 10 fold cross-validation, 2) trained on data from one trial, and then validated on data from a second, independent trial. The worst features are the features with lowest predictive power from Figure 3.2.*

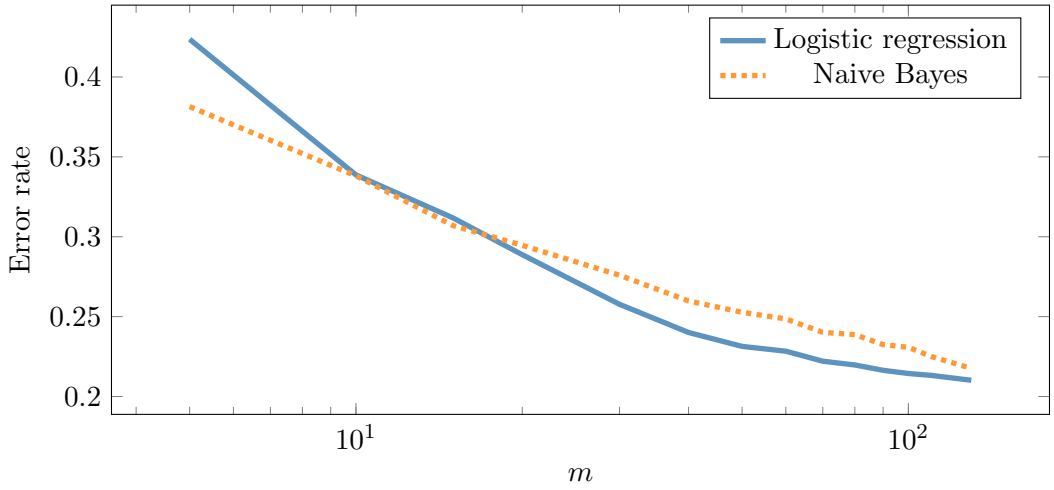


Figure 4.5: *Classifier accuracy for naive Bayes and logistic regression when successively adding more training samples. m is the number of samples of each class. The classifiers are tested on the data not used for training and each point is an average of 100 re-samples.*

4.4 Scaling to large data sets

A limiting factor in training and choosing the classifiers has been that the amount of data available has been small. Small data sizes decreases the accuracy when using the classifier on new data, since overfitting is more probable[24], which can be seen from Figure 4.5 where the validation error is shown as a function of the number of training data. Since the accuracy does not reach its asymptotic ideal performance, there is still some degree of overfitting showing that to reach the optimal performance more training samples are needed.

There are essentially two ways of countering the overfitting, and both boil down to increasing the proportion of training samples over the complexity of the classifier. So either the number of data points has to be increased or the complexity of the model must be decreased. An example of the latter would be the previous section where the complexity was decreased by decreasing the number of inputs.

If a small data set calls for a simple model, a large data set conversely allows for more complexity. And since a more complex classifier can catch a more complex behaviour[4], with for example dependencies between variables.

4.5 Robustness to mislabelled samples

Since all the data used in this thesis is labelled manually there is a risk of a few samples being mislabelled. Therefore it is interesting to see how deliberately mislabelling samples affect the accuracy of the classifiers. And as can be seen from Figure 4.6 the classifiers seem to handle small amount of mislabelled data well. As expected, the classifier becomes essentially useless when half of the data is mislabelled, returning close to random results. And when all samples are mislabelled the classifiers goes to the inverse classifier.

Especially the logistic regression classifier handle the mislabelled samples well, losing less than 1 % accuracy when 15 % of the data is mislabelled. This shows the robustness of the algorithms to having a few mislabelled samples.

[4]It is for example known[25] that neural networks with enough hidden neurons are universal approximators.

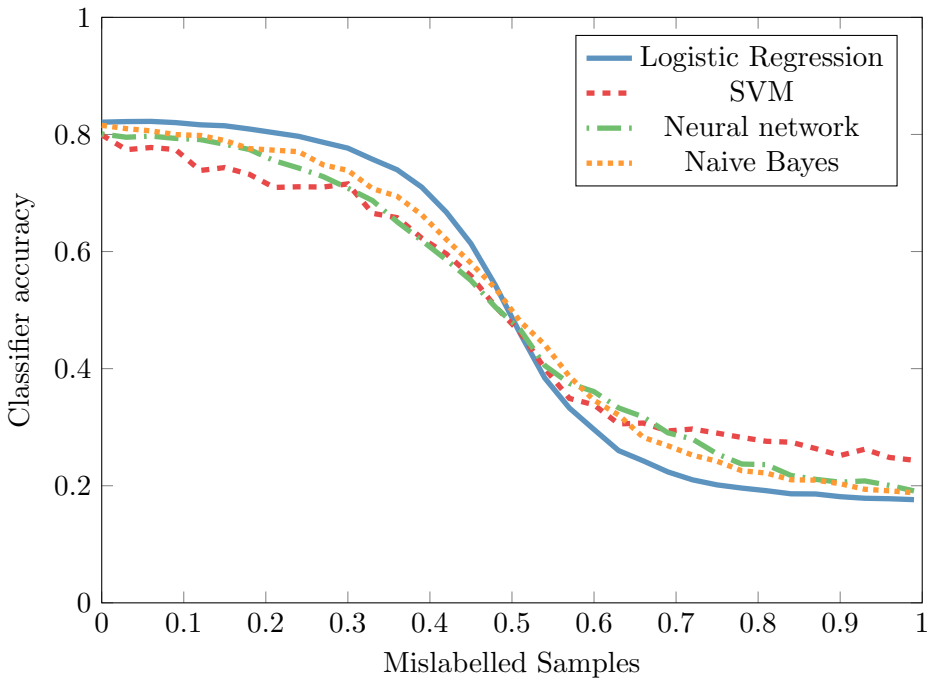


Figure 4.6: *Classification accuracy with 10-fold cross-validation when training samples are mislabelled. The implementation of the SVM algorithm started to have convergence problems when approximately 20 % of the data became mislabelled.*

5 Discussion

In this thesis it has been shown that using features extracted from the time series of tracked radar targets for classification is feasible. However, the accuracy gained this far is lower than what would be required in a true application, therefore some ways of improving the result will be discussed.

5.1 The choice of classification algorithm

The four different algorithms that were tested provided very similar result when trained and cross-validated on the same set. Even logistic regression, the most simple of the classification algorithms, performed as well as the other, more complex, methods. This hints that the limiting factor in the classification is not the classifier.

The largest difference in performance was seen in training and testing the algorithms on different sets, where naive Bayes performed worst. Using the kernel method for approximating the probability distribution gives a high dependence on the exact data in the training set. So if the data is not representative of the complete sample space, it is only to be expected that the classifier becomes overfitted. A linear classifier, such as logistic regression, would on the other hand not suffer as much from not missing some information, since its complexity is much lower.

5.2 Improving the classification results

As already mentioned, it seems that the choice of classifier does not change the performance of the classification significantly. Rather, it seems as the problem is that the information stored in the features is too low.

The question then is of course if this is the maximum attainable accuracy with the current radar system, or if too much information is discarded in creating the features. Since the first case is uninteresting, the focus will be on how to improve the features.

The first alternative is to continue on the same course, i.e. to manually chose new features to test. The problem with this, as with all manual tasks, is that it requires much time. It is also easy to miss some important feature because it is thought of as unimportant, and not even considered.

Right now the features are calculated on the data that the tracker software outputs, rather than the raw data that the radar collects. Since the tracker software outputs estimated values, some of the information stored in the initial inputs will be lost. Therefore, a plausible way forward in using manual features would be to calculate them directly on the data the radar collects, or to use the information stored in the estimation errors, i.e the difference between estimated and measured values.

The second alternative is to go back to using the raw time series as input, even though that has its problems as discussed in Section **??**, since that will prevent loss of any information. The approach called deep learning, which usually uses a variant of neural networks with many layers and neurons, have lately shown a good performance on other time series classification tasks, such as speech recognition[26]. In these, no features are extracted, this is left to the training algorithm. And as a complex enough neural network is an universal approximator[25], they are able to find any feature that a human could conceive given enough training. However,

being complex classifiers they also require large amounts of training data, making the approach unfeasible in this problem if not a large amount of tests are performed.

5.3 Representative capacity of the training data

Because of the low amount of training data it is not unreasonable to believe that the samples available does not show a complete picture of how the classes behave. The results from training and classifying on data from the first and second trial respectively seem to indicate that there indeed is a problem with how well the data spans the feature space. If no new behaviour were seen in the second trial the results of Figure 4.1 and 4.3 should be similar, which is not the case. That is, if the underlying distributions were the same in the two sets, there should be no significant difference between the second data set and one of the 10 partitions of the first data set used for cross validation, and thus the results should be similar.

But since there is a significant change in performance for most of the algorithms, especially on class B, some behaviour differs between the sets. Therefore, it is reasonable to assume that new tests will add even more new information and thus will be needed to make the classifiers more general.

5.4 The meaning of the features

The two features that performed the best were low frequency component of the estimated target size and the autocorrelation of the lateral acceleration, feature 4 and 13. The latter of these is coupled to the motion of the target, especially the turns, and indicates that class B tend to stay in a turn for longer times. It does not wander as much as individuals of class A, which have less correlation in its acceleration over a timespan of 5 s.

The estimated target size is instead coupled to the actual shape of the objects as seen from the radar. A positive bonus with this is that it is much harder to imitate if trying to make a class A object behave as a class B or vice versa. With knowledge it is possible to make objects fly in a certain pattern, but there are physical limitations in constructing them that can be harder to circumvent.

Finally, it is interesting to notice the difference in linear predictive power for feature 7 and 9, where feature 7 has a higher power. Both are the variance of lateral acceleration, but the former only during curves and the latter in straight flight. This shows that segmentation has been beneficial, and that the classes behave more different in turning than during straight flight, once again showing the slightly more wandering behaviour of class A.

5.5 Concluding remarks

From the work in this thesis it is evident that there are some difference in the behaviour of the two evaluated classes when looking at their time series, and that machine learning algorithms can find these differences. In doing this, the understanding of the classes has also been increased from learning what features is most important for the classification. Also, since the linear classifier, logistic regression, works as well as the nonlinear ones, it is shown that to increase the classifier accuracy the information the information sent to the classifiers has to be improved.

A Segmentation algorithms

An interesting set of features has been how a target behave on parts of its trajectory when it describes a special behaviour, since from initial knowledge about the two classes it was assumed that the behaviour would be different on some parts of the trajectories. Therefore, some division into for example curve and straight flight had to be done. The simplest way, and the path chosen in this thesis, is too look at the heading and define a curve as a period of high angular velocity. The problem is when to start and end the period which is considered, and it is here segmentation enters the picture.

Segmentation can be approached in several different ways. Keogh et al[27] describes three segmentation methods with varying results and complexity: the sliding window, the top down, and the bottom up algorithms. In this section these will be briefly described. But first two different approaches to the stopping criterion will be introduced.

A.1 Description of the segmentation algorithms

Segmentation can be approached in several different ways. Keogh et al.[27] describes three major segmentation algorithms with varying complexity and performance: the sliding window, the top down, and the bottom up algorithms[1]. In this section these will be briefly described. But first two different approaches to the stopping criterion will be introduced.

Stopping criterion

There exist two main approaches to the goal of a segmentation[27]: either the algorithm minimises the error, defined by the user, dividing the time series into only k segments, or else it minimises the number of segments used to represent the series, while keeping the defined error under a set threshold.

In this paper, unless otherwise specified, the first method is used with k being a linear function of the time series length. Minimising the error given a set number of segments gives the possibility to set a typical time scale for events that are of interest. This makes the algorithm less sensitive to measurement errors of the system used to collect the data, or to errors of the specific variable being segmented.

A.1.1 Sliding window algorithm

The sliding window algorithm works by continuously reading in the next value of the series and grouping measurements until the error of the approximation becomes too large. When the error of the current segment surpass a set threshold the segment is split of, and a new one is started.

Since the algorithm works on-line by construction, this method is not suitable for constricting the number of segments, rather than the error, which requires a global knowledge of the data. On the other hand, since it is an on-line method it is suitable for long continuous time series, where the complexity of calculation for the other methods are too high.

[1] They also describe how to combine the sliding window algorithm and bottom up algorithm to make a better online method. But since it gives results very comparable to the original bottom up algorithm, it will not be considered in this study.

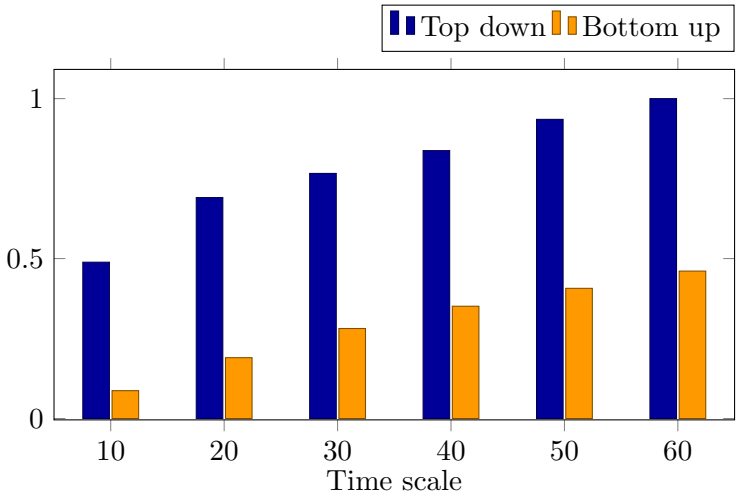


Figure A.1: *Accumulated Euclidian distance (error) between the segmentation and the heading over all the tracks in the two test sets for different values of the time scale. The values are normalised to the maximum accumulated error in the set.*

A.1.2 Bottom up algorithm

In the bottom up algorithm, the time series are divided into $N/2$ segments, where N is the number of measurements. Then the algorithm starts combining segments into longer ones until the stopping criterion is met, i.e. either using up the number of available segments, or exceeding the error.

A.1.3 Top down algorithm

Complementing the bottom up algorithm, in the top down algorithm the variable is initially assigned only one segment. Then the best point to split the segment is calculated by considering the improvement in estimation error for each possible splitting point. Once the error is small enough, or the number of segments have grown too large, the algorithm stops.

A.2 Segmentation performance

In the earlier mentioned study by Keogh et al.[27], it is found that the bottom up algorithm in general provides the best results, and sliding window mostly the worst. The top down algorithm is found to perform in between the other two algorithms, though sometimes it performs slightly better than the bottom up algorithm and at times even worse than the sliding window algorithm.

In the implementation in this thesis, it was more intuitive to use the bound on the number of segments, k, rather than on the error. Then it was possible to set a typical time scale T on which changes occur: $k = L/T$, where L is the length of the time series to be segmented. Thus, only the top down and the bottom up algorithms could be used. In Figure A.1 the performance of these two algorithms are compared for different values of the time scale T. As expected, the residual error increases as the time scale is increased. And similar to the findings by Keogh et al. the bottom up algorithm outperforms the top down algorithm on all time scales.

Even though the segmentation error decreases with smaller time scales, taking a too small T counteracts the purpose of the segmentation, giving a too fragmented segmentation. But with

a too large T the segmentation will not find all interesting features. This is illustrated in Figure A.2, where both the top down and the bottom up algorithms are used to segment the heading of a target track, for different values of T. The final T that is chosen necessarily becomes subjective, chosen where the trade-off between precision and fragmentation is "reasonable".

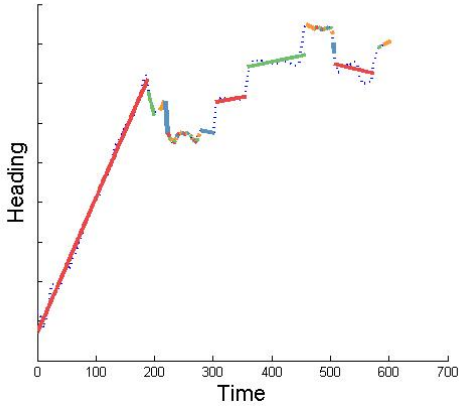
From Figure A.2 it is also seen that the top down algorithm handles step wise changes that is part of a larger trend badly. Instead of breaking the larger segment in A.2e at one of the steep changes, the algorithm breaks of small pieces at the end of the line. This is found to be typical for the top down algorithm and a major reason for choosing the bottom up algorithm for the segmentation in this thesis.

A.3 Gain of segmentation

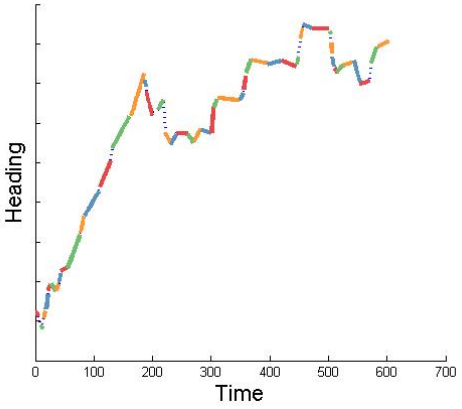
As already stated, the segmentation was performed since there was reason to believe that the two classes changed their behaviour in different ways when going from a straight path to a curve. That this is indeed true can be seen in Figures A.3 and A.4. Curves are here defined as segments from bottom up segmentation of the heading, in which the mean angular velocity is greater than 10° per minute, which was found to be a reasonable limit.

Focusing on the mean acceleration shown in Figure A.3, we see that whereas the lateral acceleration tends to be lower for class B than class A while flying straight the opposite is true in the curves. This both gives us more information about the dynamics of the two classes (B tends to make sharper turns than A), and also increases the separation between the two curves, comparing Figure A.3a and Figure A.3b.

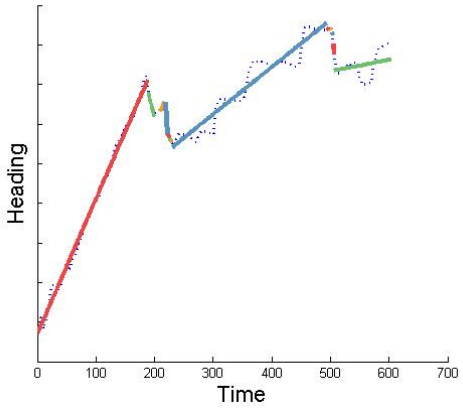
Even more difference is seen in comparing the variance of the lateral acceleration seen in Figure A.4. Here, the behaviour when the objects are flying straight is seen to be very similar and almost indistinguishable. But in the curves the variance of the lateral acceleration for class B tends to be significantly higher than for class A. Thus it is reasonable that removing the straight patches and only looking at the curves will give more information than looking at the complete path, which is indeed the case comparing Figure A.4 and Figure A.4b where the CDFs are more separated int the last figure.



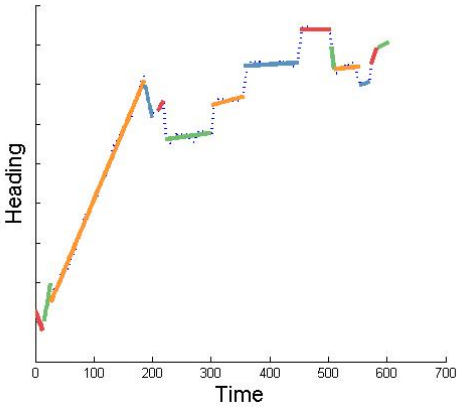
(a) *Top down*, $T = 10$



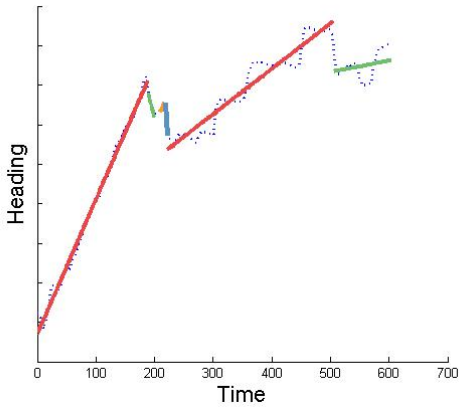
(b) *Bottom up*, $T = 10$



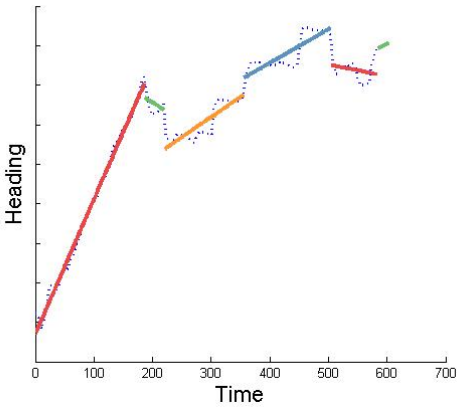
(c) *Top down*, $T = 40$



(d) *Bottom up*, $T = 40$

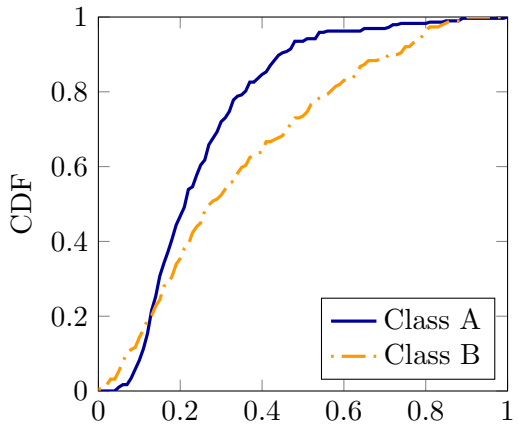


(e) *Top down*, $T = 160$

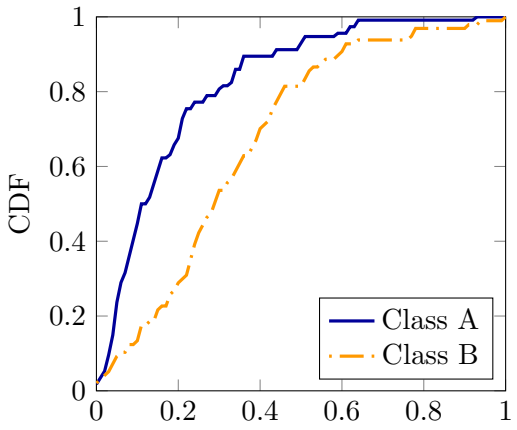


(f) *Bottom up*, $T = 160$

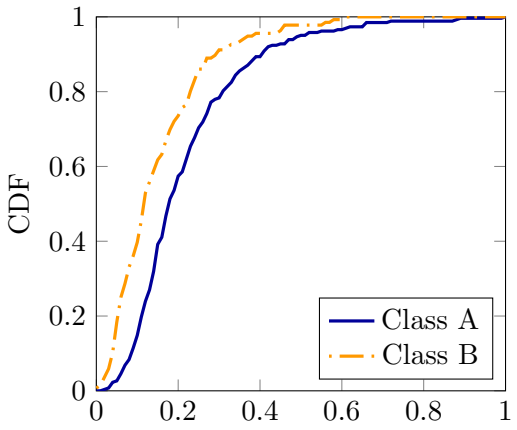
Figure A.2: *Segmentation of heading for one of the targets in the test set, using the top down and bottom up algorithms described in Section A.1, using different values for the time scale T. With too small time scales, the segmentation becomes too fragmented to be of use. But with too large T, the segmentation does not find all the changes.*



(a) *Over whole path.*

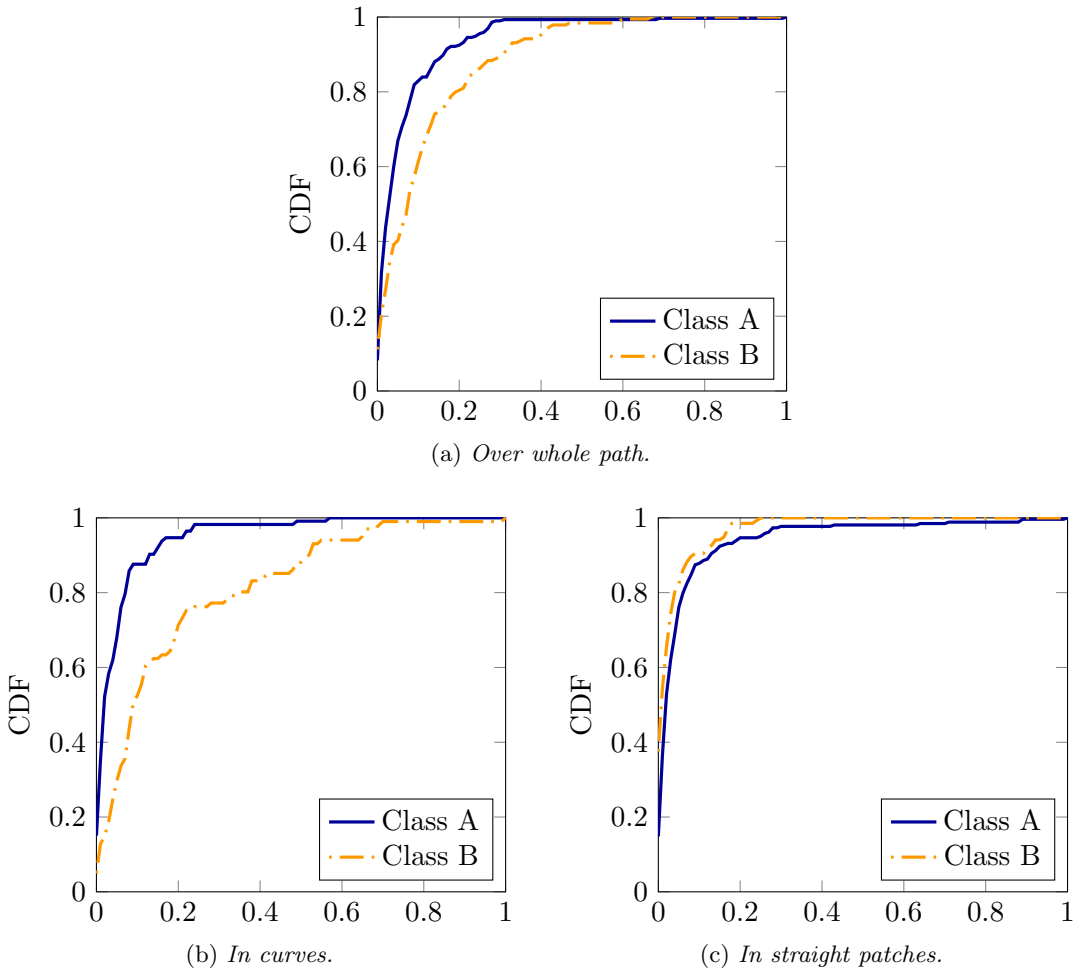


(b) *In curves.*



(c) *In straight patches.*

Figure A.3: *CDF of the mean of magnitude of the lateral acceleration, taken over the whole path, only the curves, and only the straight patches, respectively.*



(a) *Over whole path.*

(b) *In curves.*

(c) *In straight patches.*

Figure A.4: *CDF of the variance of magnitude of the lateral acceleration, taken over the whole path, only the curves, and only the straight patches, respectively.*

References

[1] G. W. Stimson, *Introduction to airborne radar*. SciTech Pub., 1998.

[2] F. Sebastiani, "Machine learning in automated text categorization", *ACM computing surveys (CSUR)* vol. **34**, no. 1 2002, 1–47, 2002.

[3] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[4] Y. Bar-Shalom and X.-R. Li, *Estimation and tracking – Principles, techniques, and software*. Artech House, Inc, 1993.

[5] D. Angelova and L. Mihaylova, "Joint target tracking and classification with particle filtering and mixture kalman filtering using kinematic radar information", *Digital Signal Processing* vol. **16**, no. 2 2006, 180–204, 2006.

[6] S. B. Kotsiantis, "Supervised machine learning: A review of classification techniques", *Informatica* no. 31 2007, 249–268, 2007.

[7] H. Zhang, "The optimality of naive bayes", *AA* vol. **1**, no. 2 2004, 3, 2004.

[8] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss", *Machine learning* vol. **29**, no. 2-3 1997, 103–130, 1997.

[9] B. Cestnik, "Estimating probabilities: A crucial task in machine learning.", *ECAI*, vol. 90, 1990, pp. 147–149.

[10] J. D. Rennie, L. Shih, J. Teevan, D. R. Karger, *et al.*, "Tackling the poor assumptions of naive bayes text classifiers", *ICML*, Washington DC), vol. 3, 2003, pp. 616–623.

[11] R. R. Bouckaert, "Naive bayes that perform well with continuous variables", *Proceedings of the 17th Australian Conference on AI*, ser. Lecture Notes AI, Springer, 2004.

[12] G. H. John and P. Langley, "Estimating continuous distributions in bayesian classifiers", *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, Morgan Kaufmann Publishers Inc., 1995, pp. 338–345.

[13] K. B. Irani and U. M. Fayyad, "Multi-interval discretization of continuous-valued attributes for classification learning", *IJCAI* 1993, 1022–1027, 1993.

[14] M. Riedmiller, "Advanced supervised learning in multi-layer perceptrons — from backpropagation to adaptive learning algorithms", *Computer Standards & Interfaces* vol. **16**, no. 3 1994, 265–278, 1994.

[15] A. Smola and V. Vapnik, "Support vector regression machines", *Advances in neural information processing systems* vol. **9** 1997, 155–161, 1997.

[16] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006, pp. 707–710.

[17] C.-W. Hsu, C.-C. Chang, C.-J. Lin, *et al.*, *A practical guide to support vector classification*, 2003.

[18] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning* vol. **20**, no. 3 1995, 273–297, 1995.

[19] Y. LeCun and Y. Bengio, "Convolutional networks for images, speech, and time series", *The handbook of brain theory and neural networks* vol. **3361**, no. 10 1995, 1995.

[20] A. Nanopoulos, R. Alcock, and Y. Manolopoulos, "Feature-based classification of time-series data", *International Journal of Computer Research* vol. **10**, no. 3 2001, 2001.

[21] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection", *The Journal of Machine Learning Research* vol. **3** 2003, 1157–1182, 2003.

[22] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms", *Neural computation* vol. **10**, no. 7 1998, 1895–1923, 1998.

[23] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection", *Ijcai*, vol. 14, 1995, pp. 1137–1145.

[24] S. Raudys and V. Pikelis, "On dimensionality, sample sizes, classification error, and complexity of classification algorithms for pattern recognition", *Pattern Analysis and Machine Intelligence, IEEE Transactions on* no. 3 1980, 242–252, 1980.

[25] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural networks* vol. **2**, no. 5 1989, 359–366, 1989.

[26] F. Seide, G. Li, and D. Yu, "Conversational speech transcription using context-dependent deep neural networks.", *Interspeech*, 2011, pp. 437–440.

[27] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "Segmenting time series: A survey and novel approach", *Data Mining in Time Series Databases*, ser. Machine Perception and Artificial Intelligence, Singapore: World Scientific Publishing, 2004.