This chapter introduces concepts and definitions that are used throughout this thesis. We begin with a short introduction to machine learning and more specifically to kernel methods. Then we give an overview of the research and necessary concepts in spatio-temporal data-mining. We end with a description of the tasks of a maritime safety and security system and the specific datasets of vessel trajectories used in our experiments.

2.1 MACHINE LEARNING

In machine learning several different learning tasks with respect to an input space of objects $\mathcal{X}$ are recognized. The most iconic task in machine learning is a form a supervised learning, called classification. In classification the input space $\mathcal{X}$ is labeled from a set of labels $\mathcal{Y}$. A training example is then an instance $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. The goal for a learning algorithm is to find a hypothesis function, or classifier, $h : \mathcal{X} \rightarrow \mathcal{Y}$, which predicts a label from $\mathcal{Y}$ for any instance of $\mathcal{X}$. The function $h$ is an approximation of some unknown target function $t$ that describes the data optimally, thus ideally we would like to have $\forall x_i \in \mathcal{X} : h(x_i) = t(x_i)$. However, for any realistic problem, the training set is a (small) subset of $\mathcal{X}$ and contains noise, and therefore $h$ will most likely never be equal to $t$. Multiple strategies, i.e. multiple learning algorithms, exist to identify $h$, which all have different advantages and disadvantages; some handle noise better, some can deal well with skewed training sets, and others require very little training time.

The prototypical unsupervised machine learning task is clustering. In clustering we have a dataset from an unlabeled input space $\mathcal{X}$, and we construct a partitioning of this dataset into clusters $\{c_1, \ldots, c_k\}$. To construct such a partitioning, a notion of similarity between objects in $\mathcal{X}$ is required. Clustering algorithms usually have parameters that control the number of clusters that are found. For centroid based algorithms, like k-means, the parameter $k$ directly sets the number of clusters created. Density based algorithms, such as DBSCAN (Ester et al. (1996)), consider as clusters groups of objects that are close together, or densely connected. These algorithms have parameters to regulate this density, in terms of the closeness together and the number of objects required in a cluster.

Generally the task of outlier detection is considered a form of unsupervised learning, since usually the dataset that belongs to input
space $\mathcal{X}$ is unlabeled. The goal is to learn a function $h: \mathcal{X} \to \{-1, +1\}$, where the value $-1$ indicates an outlier or abnormal observation, and $+1$ an inlier or normal observation. The main reason that outlier detection is considered an unsupervised task is that there is great variation in types of outliers, and there is often no single class of outlier. Moreover, the available training data (almost) only contains normal observations, since outliers are supposed to occur infrequently. Thus, an outlier detection algorithm constructs a model $h$ of the training data, such that for (nearly) all $x_i \in \mathcal{X}$, $h(x_i) = +1$. If a new observation $x'$, fits the model well enough to be considered an inlier, then $h(x') = +1$, else $h(x') = -1$, and $x'$ is an outlier.

The structure of the objects in $\mathcal{X}$ and the models that a machine learning algorithm creates can vary widely. The most common type of object in $\mathcal{X}$ is the vector of a fixed number of features $m$: $x = \langle x_1, \ldots, x_m \rangle$. These features can be nominal or numerical. Traditional machine learning algorithms like rule learners, decision trees and Naive Bayes work on these representations (Mitchell (1997)). However, more complex types of data, like moving object trajectories, can be difficult to fit into the fixed length feature vector type of representation. In this case, methods based on similarity provide a solution. A similarity measure for complex data is often more straightforward and natural to define than a feature vector representation. Clustering and outlier detection are naturally suited to use similarity measures. For classification the $k$-nearest neighbor algorithm can be used. Kernel methods, which we describe below, are also based on similarity measures.

A possible alternative to a similarity based approach is to use a probabilistic (Bayesian) framework. Trajectories can be modeled as (non)-linear dynamic systems, using Kalman filters, hidden Markov models, or more advanced dynamic Bayesian network techniques. However, the modeling choices needed to represent trajectories as probabilistic variables are non-trivial, given that there is no fixed set of times and locations. Classification can be done by training models for each class. For a new instance we select as its class the one that belongs to the model that has the highest probability of generating that instance. Outlier detection can be done by training models for the normal data and labeling new instances as outliers when they have a low probability of being generated by one of the models. For clustering a form of similarity between probabilistic models for trajectories is needed.

We feel that a similarity/kernel based framework fits moving object trajectory data well. It is a direct approach to solving machine learning tasks in this domain. However, we recognize that other solutions can also be viable for trajectories.
2.1 Machine Learning

Kernel Methods

Kernel methods are machine learning techniques that use the notion of similarity between objects (Schölkopf and Smola (2001); Shawe-Taylor and Cristianini (2004)). The framework of kernel methods provides a unified way to handle different machine learning tasks, like classification and clustering. For each task well-known algorithms exist to solve it, given that we can define a kernel function, introduced below, between the objects in our dataset. Kernels are most often associated with the Support Vector Machine (SVM) classification algorithm (Cortes and Vapnik (1995); Schölkopf et al. (2000)). However, kernels can also be used for clustering (Dhillon et al. (2007)), and for outlier detection (Schölkopf et al. (2001)).

The key insight of kernel methods is that a number of machine learning techniques, like the SVM and the perceptron, can be redefined to use a dot-product between vectors when creating a model of the data, and that this dot-product can be replaced by a mathematically more general notion called a kernel. This is what is referred to as the kernel trick. Kernels can be defined for objects with other structures than feature vectors, like trees, graphs or trajectories, in contrast to dot-products, which are computed between same length feature vectors. Many types of kernels create an implicit mapping of objects into a higher dimensional feature space. This mapping is implicit, because the kernel function directly computes the dot-product in this higher dimensional feature space, without computing the mapping. Computing this is often much cheaper than doing the mapping explicitly, if it all, and computing the dot-product in the high dimensional space.

In order to ensure that kernel functions represent a dot-product in a certain feature space, the functions need to be Positive Semi-Definite (PSD) as defined in Definition 2.1.1. A simple example of such a function is the canonical dot-product $x \cdot x'$ or the identity function.

Definition 2.1.1. Let $X$ be a set of objects. Then a function $k : X \times X \to \mathbb{R}$ is called a positive semi-definite kernel iff $k$ is symmetric, i.e. $\forall x, x' \in X : k(x, x') = k(x', x)$ and,

$$\sum_{i,j=1}^{n} a_i a_j k(x_i, x_j) \geq 0,$$

for any $n \in \mathbb{N}$, $a_1, \ldots, a_n \in \mathbb{R}$ and $x_1, \ldots, x_n \in X$.

The assumption of positive semi-definiteness is essential to ensure convexity in the quadratic programming approach used in Support Vector Machines.\footnote{The machine learning literature often refers to the properties of positive definiteness and positive semi-definiteness interchangeably, since the difference is not of great importance for machine learning. We will always use the term Positive Semi-Definite (PSD).}
In practice, good machine learning results have also been achieved with kernels that are indefinite, i.e. still symmetric but not positive semi-definite. Some research has been done on their behavior, see Haasdonk (2005). In this thesis we will use both indefinite and PSD kernels.

Using simple kernels, more complex kernels can be constructed using different kernel closure properties. For a family of PSD kernels $k_1, \ldots, k_n$ the following holds (Berg et al. (1984)):

- the sum $\sum_{i=1}^n c_i k_i$ is a kernel, given $c_1, \ldots, c_n \geq 0$,
- the product $k_1^{c_1} \cdots k_n^{c_n}$ is a kernel, given $c_1, \ldots, c_n \in \mathbb{N}$,
- the limit $k := \lim_{n \to \infty} k_n$ is a kernel, if the limit exists.

These closure properties can be used to construct more complex kernels from simpler kernels.

A PSD kernel that we will encounter in the rest of the thesis is the Gaussian kernel, for two vectors $x, x'$ and standard deviation $\sigma$:

$$k_{\sigma}(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right). \quad (2.1)$$

Often, the $\frac{1}{2\sigma^2}$ part is replaced by just the scaling parameter $\beta > 0$.

We note that we refer to both a kernel function and the kernel matrix based on that function as kernel. However, we will use the small $k$ for functions and the capital $K$ for matrices.

**Mapping kernels** To construct kernels on more complex structures than same length vectors, different solutions have been proposed. In this thesis we make use of mapping kernels (Shin and Kuboyama (2008)). Mapping kernels are generalizations of convolution kernels as defined by Haussler (1999).

**Definition 2.1.2.** A mapping kernel on a set of objects $\mathcal{X}$ requires a map $M(x, y) \subseteq \mathcal{X}'_x \times \mathcal{X}'_y$ between two objects $x, y \in \mathcal{X}$, where each $x \in \mathcal{X}$ is associated with a finite subset $\mathcal{X}'_x$ of a common space $\mathcal{X}'$. We define the mapping kernel:

$$k(x, y) = \sum_{(x'_i, y'_i) \in M(x, y)} k'(x'_i, y'_i).$$

This kernel is positive semi-definite iff the kernel $k'$ is positive semi-definite on $\mathcal{X}'$ and the map $M$ is transitive: $\forall x, y, z \in \mathcal{X} ((x', y') \in M(x, y) \land (y', z') \in M(y, z)) \rightarrow (x', z') \in M(x, z)).$

The approach to constructing a mapping kernel is to find a common space of less complex structured objects, called $\mathcal{X}'$, for the objects in $\mathcal{X}$, such that a positive semi-definite kernel can be defined on $\mathcal{X}'$. For instance, an example of complex objects are strings, and less
structured objects are the *characters* that make up the strings. A PSD kernel is easily defined on characters, e.g., the identity function. Each more complex object \( x \in \mathcal{X} \) is associated with a finite subset of \( \mathcal{X}' \), indicated as \( \mathcal{X}'_x \). In the strings example, a string would be associated with the characters that make up the string. Next, we need a transitive map \( M \) between two \( x, y \in \mathcal{X} \). This map \( M \) is a subset of \( \mathcal{X}'_x \times \mathcal{X}'_y \). With the map \( M \) we can create a positive semi-definite kernel on \( \mathcal{X} \), using the positive semi-definite kernel on the common space \( \mathcal{X}' \), as in Definition 2.1.2. A possible transitive map for the string example would be to map each character, with position, in one string to each character in the other string.

2.1.1.1 *Support Vector Machines*

The Support Vector Machine is a method that seeks to find a binary classifier for points in a \( d \)-dimensional space, by finding their optimal linear separation. For points \( x_i \in \mathcal{X} \) belonging to two classes, \( y = +1 \) or \( y = -1 \), it tries to find values for \( w \) and \( b \) such that the function

\[
f(x_i) = w \cdot x_i + b,
\]

optimally separates the two classes. This means that the hyperplane defined by \( w \cdot x + b = 0 \) has the same distance to the closest point with class +1 as to the closest point with class −1. Figure 1 illustrates this for the 2-dimensional separable case.

The general version of this problem, i.e., for datasets that are not fully linearly separable, can be solved by minimizing

\[
\frac{1}{2}||w||^2 + C \sum_i \xi_i,
\]
with respect to the following constraints:

\[ \begin{align*}
  x_i \cdot w + b & \geq 1 - \xi_i & \text{for } y_i = +1 , \\
  x_i \cdot w + b & \leq -1 + \xi_i & \text{for } y_i = -1 , \\
  \xi_i & \geq 0 & \forall i .
\end{align*} \]  

(2.4)  

(2.5)  

(2.6)

Here, \( \xi_i \) are so-called slack variables to deal with the non-separability, and \( C \) is a user determined cost parameter to set the influence of classification errors.

In practice, Lagrange multipliers \( \alpha_i \) are added to solve this optimization problem and a dual form

\[ \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i \cdot x_j , \]

(2.7)

is introduced, which needs to be maximized, subject to

\[ \begin{align*}
  0 & \leq \alpha_i \leq C , \\
  \sum_i \alpha_i y_i & = 0 .
\end{align*} \]  

(2.8)  

(2.9)

In this dual form we see why SVMs are considered a kernel method. The dot-product \( x_i \cdot x_j \) can be replaced by a kernel \( k(x_i, x_j) \). All \( x_i \) for which \( \alpha_i > 0 \) are called support vectors. These are the vectors that “support” the separating hyperplane. Multiple options exist for extending SVMs to multi-class classifiers. For more details and an excellent tutorial see the work by Burges (1998).

**ONE-CLASS SVM** For outlier detection there exists a type of Support Vector Machine called the one-class SVM (Schölkopf et al. (2001)). A one-class SVM tries to create a model of unlabeled data. The technique tries to separate most of the data points from the origin with maximum margin, using a hyperplane. This idea results in equations very similar to regular SVMs. One-class SVMs try to minimize:

\[ \frac{1}{2} \| w \|^2 + \frac{1}{\nu n} \sum_i \xi_i - \rho , \]

(2.10)

subject to

\[ \begin{align*}
  x_i \cdot w & \geq \rho - \xi_i & \forall i , \\
  \xi_i & \geq 0 & \forall i .
\end{align*} \]  

(2.11)  

(2.12)

Here, \( \rho \) is the distance of the hyperplane to the origin, \( n \) is the number of data points and \( \nu \) is a user configurable parameter that controls the trade-off between the number of support vectors and the amount of data captured by the model. To solve this optimization problem a dual form similar to regular SVM exists, which can also be expressed in terms of dot-products and hence can make use of a kernel function.
Due to the popularity of Support Vector Machines there exist a lot of good implementations that can be used off-the-shelf. We use the library LibSVM (Chang and Lin (2001)), which is freely available and can be easily called directly from MatLab. LibSVM implements a multi-class version of the often used C-SVM, which we described above, and also has an implementation of one-class SVM.

2.1.1.2 Kernel k-means

The popular and well-known clustering algorithm k-means can be turned into a kernel based algorithm, dubbed kernel k-means, in a relatively straightforward manner (Shawe-Taylor and Cristianini (2004); Dhillon et al. (2007)). This adds flexibility to the original algorithm in terms of the possible input data. Other kernel based clustering algorithms exist as well, such as support vector clustering (Ben-Hur et al. (2001)), which does not require the specification of the number of clusters \( k \), but has parameters for controlling the neighborhood.

For a set of points \( x_i \in \mathcal{X} \), regular k-means tries to find a clustering \( C \) into \( k \) partitions:

\[
C = \{ c_1, \ldots, c_k \},
\]

such that for all \( x_i \in \mathcal{X} \), \( x_i \) is in exactly one \( c_i \subseteq \mathcal{X} \), and that minimizes the objective function:

\[
\sum_{j=1}^{k} \sum_{x_i \in c_j} \| x_i - m_j \|^2 ,
\]

where

\[
m_j = \frac{\sum_{x_i \in c_j} x_i}{|c_j|} .
\]

In kernel k-means, the squared distance \( \| x_i - m_j \|^2 \) is rewritten using dot-products, which can be replaced by kernels \( k(x_i, x_j) \) between arbitrary objects \( x_i, x_j \),

\[
k(x_i, x_i) - 2 \frac{2}{|c_j|} \sum_{x_l \in c_j} k(x_i, x_l) + \frac{1}{|c_j|^2} \sum_{x_l, x_m \in c_j} k(x_l, x_m) .
\]

In Dhillon et al. (2007), weights \( w_i \) are added for each \( x_i \) to create weighted kernel k-means, leading to the objective function:

\[
\sum_{j=1}^{k} \sum_{x_i \in c_j} w_i \left( k(x_i, x_i) - \frac{2}{|c_j|} \sum_{x_l \in c_j} w_l k(x_i, x_l) + \frac{1}{|c_j|^2} \sum_{x_l, x_m \in c_j} w_l w_m k(x_l, x_m) \right) .
\]
Using different settings for the weights $w_i$, variants of popular graph cut based clustering algorithms can be solved without doing expensive eigen value computations. For instance, the often used normalized cut (Shi and Malik (2000)) can be computed using weighted kernel k-means. Let $K$ be the kernel matrix computed by kernel function $k$. If we let $D$ be the diagonal degree matrix with $D_{i,i} = \sum_j K(i,j)$, then $w_i = D_{i,i}$ and $K' = D^{-1}KD^{-1}$, where $K'$ is the kernel matrix used in weighted kernel k-means. The advantage of the normalized cut over regular kernel k-means is that normalized cut can handle clusters of different sizes better.

To compute weighted kernel k-means the most straightforward strategy is the same as for regular k-means. This means that we start out with a random initialization of $k$ clusters. At each iteration we try to minimize the distance between the cluster centers and objects in the clusters by first assigning objects to the closest cluster and then recomputing the cluster centers, until we arrive at a stable clustering. Because different random initializations can lead to different stable partitions $C$, the kernel k-means clustering is run a number of times. The partitioning with the lowest intra cluster spread is kept as the final clustering.

2.1.2 Evaluation Metrics

In machine learning different metrics are used to evaluate the performance of different algorithms. The basic score to measure the performance of classification algorithms is accuracy. Let $\mathcal{X}_{test} \subset \mathcal{X} \times \mathcal{Y}$ be a test set and let $h$ be a trained classifier, then accuracy is

$$\text{accuracy}(h, \mathcal{X}_{test}) = \frac{\sum_{(x_i, y_i) \in \mathcal{X}_{test}} [h(x_i) = y_i]}{|\mathcal{X}_{test}|}, \quad (2.18)$$

i.e. simply the fraction (usually given as a percentage) of correctly classified instances in the test set.

In evaluating a binary classifier, such as an outlier detector, with class labels $\mathcal{Y} = \{+1, -1\}$ the measures of precision and recall are often used. Let $\text{pos}_h = \{x_i \mid (x_i, y_i) \in \mathcal{X}_{test}, h(x_i) = +1\}$ and $\text{pos}_{\text{test}} = \{x_i \mid (x_i, y_i) \in \mathcal{X}_{test}, y_i = +1\}$. Precision is

$$\text{precision}(\text{pos}_h, \text{pos}_{\text{test}}) = \frac{|\text{pos}_h \cap \text{pos}_{\text{test}}|}{|\text{pos}_h|}. \quad (2.19)$$

This is the fraction of instances that are classified as positive that also belong to the positive class. Recall is

$$\text{recall}(\text{pos}_h, \text{pos}_{\text{test}}) = \frac{|\text{pos}_h \cap \text{pos}_{\text{test}}|}{|\text{pos}_{\text{test}}|}, \quad (2.20)$$
which is the fraction of positive instances that are classified as positive. Precision and recall are often combined into the $F_1$-measure,

$$F_1(\text{pos}_h, \text{pos}_{\text{test}}) = \frac{2|\text{pos}_h \cap \text{pos}_{\text{test}}|}{|\text{pos}_h| + |\text{pos}_{\text{test}}|},$$

which is the harmonic mean between precision and recall.

In tasks where a classifier produces a ranked list of results, for instance in outlier detection, we are often only interested in the top-$n$ most results. Let $\mathcal{X}^n_{\text{test}}$ be the $n$ highest ranked positive results. Then precision@n is computed as the precision for the the set $\mathcal{X}^n_{\text{test}}$, which gives the fraction of true positives among the top-$n$. Note that the definitions above can easily be applied to the negative class, which we are interested in in outlier detection, when we switch $+1$ and $-1$.

To evaluate the quality of a clustering compared to a gold standard we use a scoring function that takes the best $F_1$-score for each cluster $g_i$ in the gold standard $G$ and average over these scores (Liao (2005)). Thus the clustering score function $\text{clus\_score}$ of a clustering $C$ of size $k$ with respect to a gold standard $G$ of size $k$ is

$$\text{clus\_score}(C, G) = \frac{1}{k} \sum_{g_i \in G} \max_{c_j \in C} F_1(c_j, g_i).$$

## 2.2 Spatio-Temporal Data-Mining

The field of machine learning/data mining that is concerned with moving object trajectories and other types of spatio-temporal data is called Spatio-Temporal Data-Mining (STDM). STDM is a relatively young area of research for which there have been a number of workshops. For instance the series of Workshops on Spatial and Spatio-Temporal Data-Mining (SSTDM), such as the one in 2010 as part of ICDM 2010 (Fan et al. (2010)). Work in the STDM area builds upon work in Spatial Data-Mining and Temporal or Time-Series Data-Mining. One particular problem that results from the infancy of the field is the lack of standard datasets for testing new methods. A good overview of the research that has been done until recently is given by Nanni et al. (2008).

### 2.2.1 Moving Object Trajectories

Research in STDM has a strong focus on solving typical machine learning/data-mining tasks, such as clustering, pattern mining and classification, for spatio-temporal data. The most common type of spatio-temporal data is that of the moving object trajectory. Moving object trajectories are continuous movements in a geographical space. This continuous movement is sampled using Global Positioning System (GPS) sensors or similar techniques. In the following we will
mostly consider a trajectory from the sampled perspective, i.e. as a sequence of temporally labeled positions. A difference is made between unconstrained trajectories, e.g planes and animals, and constrained trajectories, e.g. cars that follow a road network.

We define a moving object trajectory formally in Definition 2.2.1.

**Definition 2.2.1.** A moving object trajectory $T$ in 2-dimensional space is represented by a sequence of vectors: $T = \langle x_1, y_1, t_1 \rangle, \langle x_2, y_2, t_2 \rangle, \ldots, \langle x_n, y_n, t_n \rangle$, where $x_i$ and $y_i$ represent the position of the object at time $t_i$ and $t_{i+1} > t_i$. The length of a trajectory, i.e. the number of vectors, is denoted as: $|T|$. Furthermore, let $T(i) = \langle x_i, y_i, t_i \rangle$ and $T(i, j) = \langle x_i, y_i, t_i \rangle, \ldots, \langle x_j, y_j, t_j \rangle$.

The sample rate of trajectories is not always fixed, thus the difference between consecutive values $t_i, t_{i+1}$ is not the same. In some tasks, there are more dimensions to trajectories that can be derived from the $x, y, t$ information that we need to consider, such as speed and direction. These dimensions can just be added to the $\langle x, y, t \rangle$ vector as extra variables. In the following we refer to a vector $\langle x_i, y_i, t_i \rangle$ as a trajectory element or point. In the analysis of trajectories the concepts of *stop* and *move* are essential (Spaccapietra et al. (2008)). What should be considered stops and moves is application dependent. A *stop* is the non-empty time interval for which the traveling object does not move from the application’s perspective, the rest of the trajectory is considered a *move*.

The approaches to data-mining for moving object trajectories can be partitioned into two broad types (Nanni et al. (2008)). On the one hand we have the approach that focuses on using standard data-mining/machine learning techniques and applying these to trajectory data. This requires defining features or similarity measures, depending on the technique, for trajectories. In this approach, all knowledge about trajectory data is put into the features/similarity measures. On the other hand there is the type of data-mining for trajectories that has the aim to specifically design methods for this type of data, using the specific properties of trajectory data in the machine learning algorithms.

### 2.2.2 Trajectory Similarity Measures

Due to the nature of trajectories being different in temporal length, distance traveled and the number of data points, using a feature based approach is not so obvious. Often similarity measure based techniques are used for trajectories, for which there are a number of possibilities. Note that we use the term ‘similarity’ generically, for any measure that expresses some form of ‘sameness’ between trajectories. Thus, we use it for ‘distances’, where a larger value means less equal, ‘similarities’, where a larger value means more equal, and ‘dissimilarities’, where larger values mean less equal.
One approach is to consider a trajectory as a sequence of points. For two trajectories we search for an optimal alignment between their points, according to some scoring scheme. How this alignment is determined and the scoring scheme is defined leads to different similarity measures. For instance, there exists: Dynamic Time Warping (DTW) (Vlachos (2004)), and various forms of Edit Distance (ED), such as edit distance with real penalties (Chen and Ng (2004)), edit distance on real sequences (Chen and Özsu (2005)) and Longest Common SubSequence (LCSS) (Vlachos et al. (2002, 2005)). The DTW similarity measure has its origin in the time-series literature, whereas edit distance measures were originally defined for strings of characters. We will treat these methods more extensively in Chapter 4.

Instead of looking at a trajectory as a sequence, the similarity measures considered by Nanni (2002); Nanni and Pedreschi (2006); van Kreveld and Luo (2007); Frentzos et al. (2007); Buchin et al. (2009) interpret a trajectory as a continuous function and take as a similarity measure the integral over t for the distance function d that gives the distance between two trajectories for each time point t. The measure given by Buchin et al. (2009) generalizes previous versions of this measure with both a variable time-shift and a variable length of matching. Pelekis et al. (2007) describe a different method that calculates the surface between trajectories, instead of calculating the integral over time, projections to the xy- and t-plane are used.

The key difference between alignment like approaches and the integral based approach is the treatment of time. In alignments, time is often treated implicitly, i.e. only by the ordering of the sequence elements, whereas it is treated explicitly in the integral approach.

2.2.3 Data-Mining Tasks for Trajectories

With similarity measures such as those described above, typical tasks like clustering and classification can be performed.

For clustering, Vlachos et al. (2002) take an approach using longest common subsequence combined with a density based algorithm. An integral over time similarity measure is used by Nanni (2002) with k-means clustering, and with density based clustering by Nanni and Pedreschi (2006). Trajectories are first converted into a grid based representation and then clustered using fuzzy c-means by Pelekis et al. (2009), which is an approach not directly based on similarity measures. The same holds for the framework by Lee et al. (2007), which uses segmentation of trajectories based on the minimum description length principle first, and then clusters the segments as subtrajectories using density based clustering. The somewhat older work by Gaffney and Smyth (1999) employs a very different mixture of regression models method.
Very little literature exists on classification of moving object trajectories. Vlachos et al. (2005) use a one-nearest-neighbor classifier with the LCSS similarity measure. The main use for this classifier is to test the power of the measure; the actual results for the task are of less importance. Using the partition-and-group framework developed by Lee et al. (2007) a classification method for subtrajectories is defined by Lee et al. (2008b). A voting algorithm for segments of trajectories is introduced by Panagiotakis et al. (2009) to classify trajectories into their representative paths.

Also, there is not much work on outlier detection for trajectories. With the partition-and-group framework from Lee et al. (2007, 2008b) an outlier detection method for subtrajectories is introduced by Lee et al. (2008a). Bu et al. (2009) define an outlier detection algorithm for trajectory streams which uses a Euclidean distance between trajectories, where the main focus is to provide an efficient datastructure to perform outlier detection online.

The prototypical data-mining task of frequent pattern mining has received a lot of attention in the spatio-temporal research. For instance, Giannotti et al. (2007) introduce a method to mine so-called T-patterns from trajectories, which is based on an existing pattern mining technique. T-patterns are concise descriptions of frequent behaviors in terms of both the regions visited during movements and the duration of the movements. Instead of searching for any type of frequent spatio-temporal pattern there is also research on finding specific patterns. For instance, this includes the search for spatio-temporal group patterns, such as, flocking, convergence and divergence (Andersson et al. (2008)). An obvious task in the context of moving objects is that of prediction, i.e. discovering where a moving object will go next. One option is to employ clustering or sequential pattern mining models for this task. For more on these types of research, see the chapter by Nanni et al. (2008).

Closely related to the field of spatio-temporal data-mining is the field of visual analytics for trajectory data. Techniques for visualization as described by, for instance, Andrienko et al. (2007, 2009), make extensive use of data-mining methods for moving objects, especially clustering.

2.2.4 Temporal Data-Mining and Kernel Methods

Moving object trajectories are a special type of time-series. Temporal data-mining has been around longer and there exists a large body of literature on it. The typical data-mining tasks mentioned above have all been studied for time-series. Some of the similarity measures mentioned earlier have their origin in time-series research. A large part of the time-series studied in temporal data-mining are rather different from moving object trajectories, e.g. financial time-series, electro car-
diagrams, etc. Typically, the objects studied are 1-dimensional time-series. For a recent overview see the paper by Fu (2011), or the less recent work by Antunes and Oliveira (2001); Roddick and Spiliopoulou (2002). Liao (2005) gives an overview of clustering methods for time-series data.

Kernels for Time-Series To the best of our knowledge, kernel methods have not been applied to tasks on moving object trajectories. However, there exists a body of work on kernels for other types of time-series. Joder et al. (2008); Gudmundsson et al. (2008); Chaovalitwongse and Pardalos (2008) use kernels based on dynamic time warping (DTW) in time-series classification. The regular form of DTW is used by Gudmundsson et al. (2008); Chaovalitwongse and Pardalos (2008), whereas Joder et al. (2008) use DTW-kernels based on definitions given by Cuturi et al. (2007), which are guaranteed to be positive semi-definite. Kernels based on the longest common subsequence are used for classification by Gruber et al. (2006).

Kernels that are not based on alignments are also used, e.g. Rueping (2001) introduces some elementary time-series kernels. Sivaramakrishnan et al. (2007) define a kernel based on linear combinations of piecewise polynomial functions. Other types include the autoregressive kernel (Cuturi and Doucet (2011)) and the cross-correlation kernel (Wachman et al. (2009)). An edit distance based kernel is used by Chandrakala and Sekhar (2008) for density based clustering of time-series.

2.3 Maritime Safety and Security Systems

The goal of the Poseidon project, the project in which the research in this thesis was conducted, was to create a Maritime Safety and Security (MSS) System. Such a system has the task to acquire, store and analyze data and information that are collected from various sources and enable users to analyze and operate on this information. The sources of information available in the project were vessel movement data obtained via the Automatic Identification System (AIS, introduced below), and historical information about vessels and geographical domain knowledge extracted from the Internet. Furthermore, an MSS system will likely include radar information, image and video data, which were unavailable in the project.

An important part of an MSS system is the automatic analysis of vessel movements stored by the system. The analysis of these trajectories can be further enhanced by integrating information from other sources such as geographical domain knowledge about harbors, anchoring areas, etc. Different types of analysis are possible that lead to different types of information. To gain insight into different vessel behaviors, groups of similar movements need to be created, which
is a clustering task. These unlabeled groups can be assigned meaningful labels by hand. Predicting known labels/properties, such as the vessel type, is another form of analysis, which is a classification task. Identifying strange trajectories, i.e. showing irregular behavior, among a large group of normal trajectories is an outlier detection task. More analysis tasks are possible. However, these three tasks represent three distinct and often occurring possibilities. For each of these tasks we have created experimental datasets, which we introduce below.

There exist some very specific research on detecting anomalies in vessel trajectories tracked with the Automatic Identification System. This work takes a different approach than this thesis, it tries to create probabilistic models on the basis of individual AIS position messages, i.e. they do not consider trajectories. Ristic et al. (2008) present a method using kernel density estimation, whereas Laxhammar (2008) introduces a Gaussian mixture model approach. Laxhammar et al. (2009) compare both methods and the kernel density estimation technique is found to be slightly better.

2.3.1 Vessel Trajectories from AIS

The vessel trajectory data that we use for the experiments in this thesis are collected using the Automatic Identification System (AIS). An AIS system is mandatory for vessels over 300 metric tons. The system broadcasts a number of different types of messages. The most often sent messages are the dynamic messages, with updates on position information, every couple of seconds, and the static messages, with static properties of the vessel, every couple of minutes. The dynamic message contains fields like a unique vessel identifier, the MMSI number, the position of the vessel in latitude, longitude format, the vessel’s speed and a timestamp. In the static message information about the vessel’s size, callsign and type is transmitted. Position of the vessel is most often determined using the vessel’s onboard GPS. These vessel trajectories are a type of unconstrained trajectories, though vessels moving in a harbor are sometimes considered to be constrained and even at open sea, more and more obstacles appear each year.

To compute distances between two latitude, longitude positions \((\phi_1, \lambda_1), (\phi_2, \lambda_2)\) one can use the haversine formula:

\[
2 \arcsin \left( \sqrt{\sin^2 \left( \frac{\phi_1 - \phi_2}{2} \right) + \cos(\phi_1) \cos(\phi_2) \sin^2 \left( \frac{\lambda_1 - \lambda_2}{2} \right)} \right)
\]

(2.23)

However, compared to taking a straightforward Euclidean norm between two \((x, y)\) vectors, this is a slow computation. Thus to speed up
most computations in the rest of this thesis, we project latitude, longitude coordinates \((\phi, \lambda)\) to a 2-dimensional \(x, y\)-plane using a simple sinusoidal projection:

\[
\begin{align*}
    x &= (\lambda - \lambda_0) \cos(\phi), \\
    y &= \phi - \phi_0.
\end{align*}
\]  

where \((\phi_0, \lambda_0)\) is the center of our dataset. Using this sinusoidal projection we get moving object trajectories as in Definition 2.2.1. More complicated projections are also possible, but this projection is sufficient for the size of the areas that our datasets are concerned with. For our datasets, the sinusoidal projection leads to differences of less than 0.5% in distance computations at the edge of the area. For analysis on datasets that cover larger areas, or areas closer to the poles, this simple projection might lead to problems and more sophisticated projections can be necessary.

Using the two-stage compression algorithm described in Chapter 3 we cut the trajectories at the stops for each of the datasets that we describe below. Thus the trajectories that we consider are moves in terms of the stop-move model by Spaccapietra et al. (2008). Moreover, we do not allow appearing or disappearing trajectories. This means that we only keep trajectories that start from a stop, or enter our area of observation and that either stop somewhere, or exit our area of observation. For all trajectories \(t_1\) is set to 0, thus we consider only the relative time of trajectories.

### 2.3.2 Clustering

For our clustering experiments we created a dataset of a 25km radius consisting of 714 vessel trajectories around the Texel island in the Netherlands. These trajectories are partitioned into 8 different clusters, creating a gold standard \(G = g_1, \ldots, g_8\). The clusters are very different in size, ranging from 8 to 348 trajectories. The average length of a trajectory is a sequence of 300 \((x, y, t)\) vectors.

We illustrate the clustering in Figure 2. The general direction of movement for each of the 8 clusters is indicated with an arrow. These clusters are unlabeled, but they can be given descriptive labels, such as “leaving port to the north” for 3 and “from Den Helder to Texel” for 8. Together, these 8 clusters give a summary, in terms of the different types of movements/behavior, of what goes on around Texel island.

### 2.3.3 Classification

To do classification experiments we use vessel trajectories in a 50km radius around the Port of Rotterdam. This set consists of 2219 trajectories. The AIS system provides a number of different vessel types.
The type distribution in our set is very skewed. For our classification experiments we use only the four most common types: cargo ship, tanker, tug and law-enforcement vessel. The average sequence length for this set is 689 elements.

The trajectories for the four types are illustrated in Figure 3. From the figure we see that cargo ships and tankers have similar trajectories, and law enforcement vessels and tugs as well. Thus, separating cargo ships from tankers and tugs from law enforcement will be the hardest classification task. More useful classification tasks than predicting the type of vessel can be thought of, since the vessel type is already available via AIS. However, predicting the vessel type is still useful to check whether the vessel type field is filled in correctly, and hence determines the trustworthiness of the vessel. Moreover, predicting the vessel type is a good example classification task, precisely because the labels are readily available via AIS, whereas other labels are harder to come by.
2.3.4 Outlier Detection

For outlier detection we selected only the cargo ships and tankers from the classification dataset described above. In this set 39 trajectories were labeled as outliers because they show strange behavior compared to the 747 trajectories that were labeled as regular. For this set, the average sequence length is 918 vectors.

The outliers are plotted against the regular trajectories in Figure 4. From the figure we see that trajectories can be outliers for different reasons: vessels can sail to an irregular place, have extra turns, or stop unexpectedly, etc.
Figure 3: An overview of the classification vessel trajectory dataset. A solid dot indicates the start of a trajectory and an asterisk indicates the end.

Figure 4: An overview of the outlier detection vessel trajectory dataset. A solid dot indicates the start of a trajectory and an asterisk indicates the end.