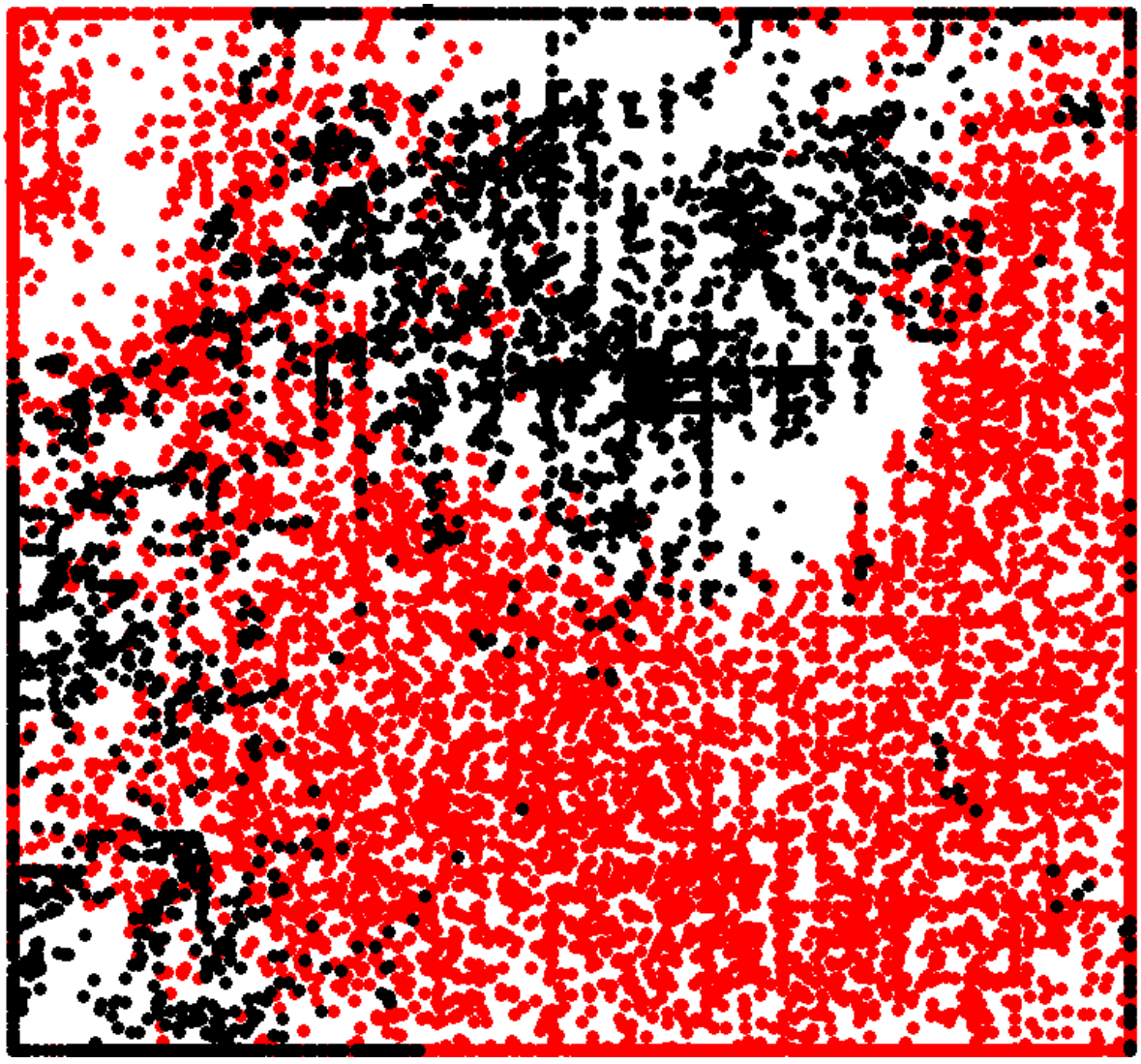


Behavior Classification based on Sensor Data

Classifying time series using low-dimensional manifold representations

Master's thesis in Engineering Mathematics and Computational Science

John Rosén

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2015
Master's thesis 2015:51

MASTER´S THESIS IN ENGINEERING MATHEMATICS AND COMPUTATIONAL SCIENCE

Behavior Classification based on Sensor Data

Classifying time series using low-dimensional manifold representations

John Rosén

Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
CHALMERS UNIVERSITY OF TECHNOLOGY

Gothenburg, Sweden 2015

Behavior Classification based on Sensor Data
Time series classification using low-dimensional manifold representations
JOHN ROSÉN

Master´s thesis 2015:51
ISSN 1652-8557
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Sweden
Telephone: +46 (0)31-772 1000

Cover:
Projection of data onto manifold warped by the Kohonen Self-Organizing Map methodology.

Chalmers Reproservice
Gothenburg, Sweden 2015

Behavior Classification based on Sensor Data
Time series classification using low-dimensional manifold representations
Master´s thesis in Engineering Mathematics and Computational Science
JOHN ROSÉN
Department of Applied Mechanics
Division of Vehicle Engineering and Autonomous Systems
Chalmers University of Technology

ABSTRACT

This master´s thesis focuses on developing and testing methods that can automatically classify a given time series as having a certain *behavior*, chosen from a set of pre-specified, known behaviors.

The first part of the thesis focused on finding statistical values where the empirical cumulative distribution of these values could be used for classification. The inverse of the cumulative distributions where then sampled at equally distanced sampling points and the resulting vector of sample values were treated as points in a high-dimensional Euclidean space. These points were then dimensionally reduced using projections onto a 2-dimensional manifold, where the manifold was warped in the high-dimensional Euclidean space using the elastic map and Kohonen Self-Organizing Map methodologies. The outputs from the manifold projections were then clustered using a k-nearest-neighbor algorithm.

Both methodologies gave fairly good classification result for the two behaviors under consideration (86.5% / 80.3%, class C_1 / C_2 for elastic map, 83.6% / 78.3%, class C_1 / C_2 for Kohonen SOM). It was also shown that there truly were convergence in distribution for the behaviors under consideration.

Key words: Time series classification, convergence in distribution, dimensionality reduction Elastic map, Kohonen SOM, k-nearest neighbors

PREFACE

This master´s thesis has been carried out at Saab Electronic Defense Systems in Kallebäck, Gothenburg, and is the final part of my M.Sc. degree at Chalmers University of Technology.

Electronic Defense Systems (EDS) is a business area within Saab group and is a world leading supplier of surveillance solutions, flight electronics and systems that can discover, localize and protect against threats.

ACKNOWLEDGEMENTS

First of all I would like to thank my supervisor Christoffer Brax at Saab EDS, for providing the opportunity to write this thesis as well as giving great support when needed. I would also like to thank my examiner at Chalmers, Dr. Krister Wolff, for providing outstanding support and guidelines for the thesis.

In addition, I would like to thank all my wonderful colleagues at Saab EDS for providing laughter, friendship and a sense of compassion and responsibility.

My final thanks goes to the founders and/or creators of Matlab, Microsoft Paint, the color black and the inventor of the table-function in Microsoft Word.

Göteborg June 10, 2015
John Rosén

CONTENTS

1 Introduction

1.1 Background

With the aid of sensor systems it is possible to measure properties of a wide range of physical objects. Working with radar systems, such properties may include position, velocity and an apparent *size* of the object (e.g. the amount of reflected radar energy). All these measured properties taken together over time may be denoted as the *behavior* of the target.

Examples of such behaviors could be commercial flights landing/taking off from airports or following flight corridors, but it is possible to define any number of different behaviors arbitrarily. However, only a handful of these will have practical use, such as the examples given. The true number of needed behaviors is for this reason quite small. The real challenge is to classify any observed target behavior to a known set of pre-specified behaviors.

The need for an applied classification methodology that can do this task is to make classification independent of whether the object divulges its own intentions; It will be able to classify behaviors regardless of any interactive communications - as well as finding *anomalous* behaviors (behaviors that cannot be easily classified to the set of pre-defined behaviors).

In general, any behavior classification methodology will provide great support for a (human) radar operator as it will highlight certain behaviors (or anomalous such) and can find these behaviors among a large number of radar objects simultaneously. One real-world example where such classification would have been of great use is the hijackings during the $9/11$ attacks.

1.2 Purpose/Objective

This thesis aims at finding, implementing and evaluating different models that can classify the behavior of an object based on sensor data.

To be considered successful, the following constraints should remain unbroken:

1. One minute of observations should be sufficient for classification
2. The model should have low dependence on the direct numerical values of the sensor data
3. The model should be able to use different lengths of observational data for classification
4. In real-time operation, the model should be able to classify 1000 objects each second using limited amounts of computational resources (e.g. running on a laptop)

The 1-minute observational constraint is set in accordance to the longest acceptable period for an object to remain unclassified. Longer periods would likely be better in terms of correctly classified objects, though, requiring such observational lengths would defeat the purpose of the intended classifier.

Being independent of the direct numerical values means that the model, for example, should be independent of the direction, position or velocity in terms of absolute values – what is only of interest are the *relative* values. This would be equivalent to the model using the subsequent *changes* in the direct values or some transformation removing the direct dependencies altogether.

The different-length constraint is due to the method in which the data is collected. In real-time operation, it will be a continuously increasing time-series and any model should be able to use *all* given data, preferably without the need of using different system parameters for different time-series lengths.

The last constraint, regarding the capacity of the model, is set to ensure that any plausible scenario will not result in system overloading – classification should be possible even given a fairly large set of radar objects. 1000 objects is well above the common number of objects observed, thus setting a large margin for computation.

1.3 Limitations

Classification of time series is often a truly nontrivial matter. This is especially true for this project as there likely is no single easily calculated quantity that can accurately classify the given set of behaviors; It is the evolution of the observed properties over time that distinguishes them. Some simplifications are therefore introduced to make data processing and classification feasible:

1. Each given time series, no matter what original length, is split into individual, shorter time series that may have some overlap. Yet, each segmented time series is treated as an independent observation.
2. Each time series may originally contain more than one behavior at different times. However, this overlap of behaviors is ignored and each segmented time series is considered to have only one corresponding behavior.
3. Due to the sensor system itself, the given data contains lots of holes (i.e. missed updates) as well as uncertainties in some of the measured properties. These holes are filled using simple linear interpolation and the measurement uncertainties are not considered at all.
4. A fully functional classifier should be able to handle an arbitrary set of pre-specified behaviors. However, only two main behaviors are considered in this project. These are denoted as C_1 and C_2. Class C_2 is sometimes regarded as a *superclass* to a set of subclasses, denoted as C_{2a}, C_{2b}, C_{2c}, C_{2d} and C_{2e}, but any classification result for C_2 is the mean of the classification result for all the subclasses.

1.4 Reading guidance

The next chapter, Chapter 2 (*Theory*) gives a short description of some common notations, algorithms, clustering techniques and general concepts encountered in this thesis, described both from a general point of view as well as mathematical when needed.

Chapter 3 (*Data structure and methodology*) contains descriptions of the given data, the methodology in which this data is processed and the statistical values derived from the data. It also contains descriptions of some commonly encountered problems when working with high-dimensional data such as the one given and motivates the chosen way in which the data is processed.

In chapter 4 (*Computational complexity*) the required computational resources of the intended classification methodology is analyzed, both in the context of model parameter estimation as well as real-time operation.

The result of the classification is presented in Chapter 5, both as the result from the "main" model setup but also the result obtained when changing some model parameters. The experimental setup is described first.

In Chapter 6 and 7 (*Discussion and conclusion* and *Future work*) conclusions are drawn from the result and some suggestions for future work are given.

1.5 Related work

There exist many different classification methodologies that can be used to classify time series data, such as *Dynamic Time Warping* and related methods. However, it is suspected that this method have a lot of drawbacks if the constraints are to be fulfilled, especially the constraint regarding the independency of direct numerical values (e.g. an classified behavior should always be the same regardless if the trajectory of the object is rotated around any axis). Classification of time series using the distribution of some observed (and further processed) properties is used instead.

A method that was used with fairly good results is the elastic map methodology applied to the problem of identifying the correct flow regime in an air-water pipe flow based on differential pressure measurements[1]. In this case, many different classification methodologies were tested and it was shown that the elastic map method outperformed the others. But more importantly so, this work showed that it was truly possible to classify time series using (non-parametric) distributions.

Another method used with good result when applied to the problem of estimating stock prices is Kohonen self-organizing map (Kohonen SOM)[2]. In this case, stock data (e.g. price, volume) for the previous 65 days were used to predict the stock price for the following day. This method did not use the distribution of the time series for classification, but it showed that self-organizing maps can be used for time series prediction (in practice, classification) with fairly good accuracy.

2 Theory

2.1 Key features

2.1.1 **Metric**, **Distance metric** or **Distance function**

A metric, or distance function [3], is a function that introduces the notion of *distance* between objects in a set. It is a function that for any two objects in a set returns a positive scalar value, representing the distance between the objects. If the main set is denoted by X, in mathematical terms the distance function is defined as

$$d: X \times X \to \boldsymbol{R}$$

For any objects x_1, x_2 and x_3 in X, the following properties must apply for a distance function d:

1. $d(x_1, x_2) \geq 0$ (non-negativity)
2. $d(x_1, x_2) = 0$ iff $x_1 = x_2$
3. $d(x_1, x_2) = d(x_2, x_1)$ (symmetry)
4. $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$ (triangle inequality)

In this thesis, the notion of *norm* is treated as equivalent to a distance function. Norms are denoted as $\|\cdot\|$.

2.1.2 Manifold

In mathematical terms, a manifold [4] is defined as a topological space that resembles the Euclidean space near each point in the manifold. Specifically, in the local neighborhood around any point in an n-dimensional manifold there exists a homeomorphism to the n-dimensional Euclidean space, meaning that there exists a continuous, bijective (one-to-one) mapping between the manifold and the Euclidean space

One example of a manifold is Earth's surface, where the surroundings around any point on the surface can be appropriately mapped by a 2-dimensional chart. Another good example is to crumple up a piece of paper, where the paper, if intact, can be stretched to its original form. This stretching of the paper represents the continuous mapping between the "crumpled up" space (manifold) and the straight space (Euclidean)

2.1.3 Expectation-Maximization algorithm

In its original setting, the expectation-maximization (EM) algorithm [5] is an iterative method used in statistics to find the maximum likelihood estimates for latent (i.e. hidden or inferred) model parameters. However, in this project the notion of usage of the expectation-maximization algorithm will not necessarily be related to maximum likelihood estimates (or statistics for that matter) but will refer to the iterative process employed in the EM-algorithm for finding local extrema.

In general, problems for which the EM-algorithm is suitable are those where data points are to be clustered in separate classes in such a way that the summarized "fitness" for all data points of a given set (i.e. how well each data point belongs to its given class) is to be maximized. Typically, having the objective of finding the global optimum is an NP-hard problem (as all different clustering settings must then be evaluated) but the EM-algorithm provides an efficient method of finding at least a local optimum.

The algorithm starts with assuming some clustering for all data points. In the **expectation** step, all parameters associated with any certain class are optimized given the data points that belong to this class and with respect to some specified fitness function (i.e. a measure of how well the entire system (parameters and clustering) is optimized). This is applied for all classes while the clustering is kept unchanged. Then, in the following **maximization** step, the fitness between any given data point and the given classes is calculated and the data point is clustered to the class where the fitness is the largest while the class parameters are kept constant. Most importantly, the fitness is calculated using the same measure as in the expectation step. With a new clustering given, the algorithm is repeated until there is no change, after which it is terminated.

The usage of the same fitness measure in both the expectation and maximization step is what guarantees a convergence to at least a local optimum. The reason is that in both steps, the summarized fitness can never decrease and thus after each iteration the clustering will always be better or unchanged with respect to this fitness measure.

2.1.4 k-means clustering

k-means clustering [6] is a clustering method which aims at dividing n data points (from the same metric space) into k classes, where each data point belongs to the cluster with the nearest mean, thus partitioning the data points into k Voronoi cells.

It is not clear from this description which type of distance measure that is to be used. The standard Euclidean distance metric is mostly used in the literature (and then sometimes used in the main definition of the method itself), but the main purpose of this method does not exclude the usage of some different distance metric. However, it will henceforth in this method description be assumed to be the Euclidean distance.

A great advantage of this metric is the ability to describe this method in terms of a fairly easily solved minimization problem. Assume n observations from the metric space $(X, \|\cdot\|_{l^2})$ which are to be divided into k clusters. Denote by S_j the set of observations that belong to cluster j and $\boldsymbol{S}$ the superset of these, i.e. $\boldsymbol{S} = \{ S_1, S_2, \dots, S_k \}$. The objective is then to find $\boldsymbol{S}$ such that

$$E = \sum_{j=1}^{k} \sum_{\boldsymbol{x} \in S_j} \left\| \boldsymbol{x} - \mu_j \right\|^2$$

is minimized. Here, μ_j denotes the mean of cluster S_j. The advantage of using the Euclidean distance is then made clearer since the point that minimizes the within-cluster sum of squares is equal to the mean of the cluster points, which need not be true for different metrics.

The expectation-maximization algorithm can be appropriately applied to this problem. Starting out by randomly assigning each data point a class membership, then for each class the mean is calculated. All the data points are then re-clustered to the cluster with the nearest mean, after which a new cluster mean is calculated and so forth. As such, both in the expectation and "maximization" step, the value of E will either decrease or remain constant and thus insuring convergence to at least a local minimum.

2.1.5 k-Nearest Neighbor Classification

Clustering method (commonly denoted as k-NN [7]) which assigns a class membership to any new observation based on the most numerous class among the new observations k nearest neighbors. It is very much different from k-means clustering, where the entire set of observations is clustered simultaneously. Here, observations are introduced individually and given a class membership based on the closeness to a set of observations of known classes.

The method itself is fairly straight forward. Given a set of N observations of known class, the distance between these observations and any new observation of unknown class can be

calculated given a distance metric. The distances to all the known observations may then be sorted, and the new observation is said to belong to the most common class found for the k first entries in this sorted distance list. In this setting, 1-NN corresponds to the simple *nearest neighbor*, where each new observation belongs to the single nearest observation of known class. A graphical example for 2-, 5-, and 10-NN is given below.

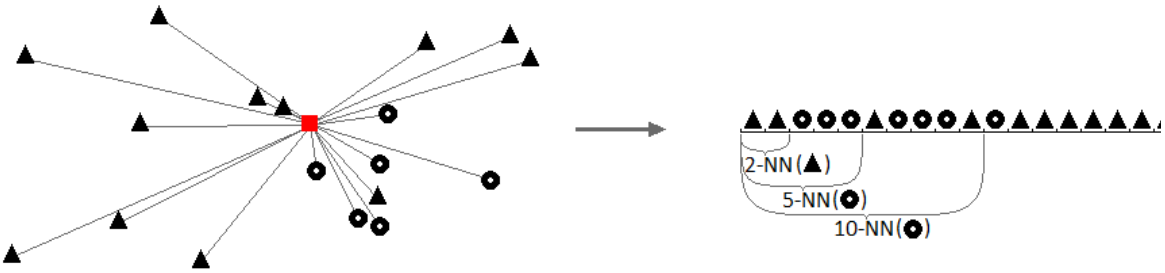


Figure 2.1: k-NN example

In the example above, the red square is to be classified either as a ring or triangle. Note that the resulting classification is dependent on how many neighbors that are considered.

2.2 Dimensionality reduction

Dimensionality reduction may generally be stated as a method of representing points of data using fewer parameters than the dimensionality of the data space[8]. This parameter space may belong to the same metric space as the dataspace does (e.g. projection in Euclidean space) or belong to some metric space where the distance metric is fundamentally different (e.g. parametrization of a curve).

Two examples of dimensionality reduction of data from a 2-dimensional Euclidean space, one linear and one non-linear, are given to the right. In the left figure it is evident that the data is heavily linearly correlated and can adequately be represented as points along a single straight line. In the right figure, the data set can also be appropriately described as points along a line, though this line is not straight – it is bent in order to better encapsulate the general shape of the data set.

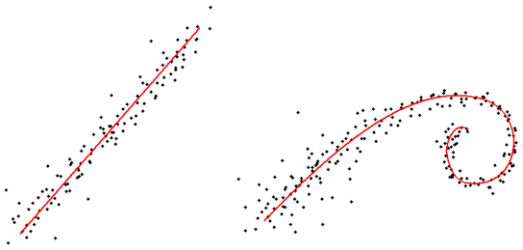


Figure 2.2: Example of dimensionality reduction

It is also clear from both figures that the dimensionality reduction need not be exact; There will be some errors in the dimensionally reduced representation. In these specific cases, the error occurs when the data points are projected onto the lines, after which the information about the original positions of the data is lost. It will rarely be the case that the data can be perfectly represented by a lower-dimensional system (if it could be, the additional dimensions would be truly redundant). The hope is that the information lost is not vital for the purpose of this project.

As previously stated, the dimensionally reduced system need not belong to the same metric space as the original system. One such example is the normal (Gaussian) distribution. The distribution itself, as a curve describing the probability of some event (as a probability density or cumulative distribution) belongs to an (uncountable) infinite-dimensional function space. The common

Euclidean distance metric is for this reason inappropriate for measuring distances between 'points' in this space and different distance metrics needs to be used instead. However, knowing that the function is normal (Gaussian), it can be parametrized using just two fundamental parameters; It's mean and variance. In this parameter space, the standard Euclidean distance metric *can* be applied to measure distances. Thus, the dimensionality is substantially reduced and the notion of distance is quite different. In mathematical terms, these spaces represent two different metric spaces.

2.2.1 PCA

Principal Component Analysis (PCA) [9] is one of the simplest ways of making a low-dimensional representation of high-dimensional data. In practice, it is an orthogonal transformation of the original and possibly linearly correlated data space into a 'new' data space of equal or lower dimensionality where the data has no linear correlation with itself at all.

Another way of describing this procedure is to say that the data is represented using coordinates of a different, new set of basis vectors. The basis vectors of this set are pairwise orthogonal (i.e. the angle between two basis vectors is always 90^{o}) but may, when represented in the original data space, have different lengths. This set of new basis vectors is thus the stiff rotation and individual scaling of the original set of basis vectors.

For PCA, the rotation and scaling of the basis vectors is chosen in such a way that the data, when expressed using the coordinates of the new basis vectors, has lost all linear correlations. The basis vector with the largest scaling then corresponds to the direction at which most variance is seen in the data and is set to be the first principal component; the basis vector with the second largest scaling is the second principal component and so forth. From this setting it is clear that the choice of principal components is dependent on relative scaling of the original data set. However, it is invariant to rotations of the data in the original data space.

The transformation may be found as follows. Denote by X the data set and C the covariance matrix of the data set, i.e. $\mathrm{cov}(X) = C$. What is to be found is a linear transformation matrix P such that $\mathrm{cov}(XP) = I$, that is, the covariance matrix in the transformed system is equal to 1 at the diagonals and 0 elsewhere.

From the general properties of the covariance matrix and the desired result we have that

$$\mathrm{cov}(XP) = P^T \mathrm{cov}(X) P = P^T C P = I \;\Leftrightarrow\; (P^T C P)^T = P C^T P^T = P C P^T = I^T = I$$

For any eigenvector v_i of the covariance matrix C we have that $C v_i = \lambda_i v_i$, where λ_i is the corresponding eigenvalue. Then if we by V denote the eigenvector matrix (i.e. $V = [v_1 \;\; v_2 \;\; v_3 \ldots]$) and Λ the eigenvalue matrix (i.e. matrix with eigenvalues on the diagonals but otherwise empty) and noting that for any eigenvector it must hold that ${v_i}^T v_j = 1$ if $i = j$ and 0 otherwise, we have that

$$CV = V\Lambda \;\Rightarrow\; V^T C V = V^T V \Lambda = \Lambda$$

One may now define the matrix $\sqrt{\Lambda}$ as the eigenvalue matrix with the square root of the eigenvalue on the diagonals and $\mathrm{inv}(\sqrt{\Lambda})$ as the inverse of this matrix, i.e. $\sqrt{\Lambda} \cdot \mathrm{inv}(\sqrt{\Lambda}) = I$ (this inverse matrix will be diagonal with $1/\sqrt{\lambda_i}$ on the diagonals). Now it is easily seen that if we set $P = V\ \mathrm{inv}(\sqrt{\Lambda})$ we get that

$$\mathrm{cov}(XP) = P^T \mathrm{cov(X)} P = P^T C P = \mathrm{inv}\left(\sqrt{\Lambda}\right) V^T \mathrm{CV}\, \mathrm{inv}\left(\sqrt{\Lambda}\right) = \mathrm{inv}\left(\sqrt{\Lambda}\right) \Lambda\, \mathrm{inv}\left(\sqrt{\Lambda}\right) = I$$

Thus we have that this new set of basis vectors is precisely the set of eigenvectors to the covariance matrix, each eigenvector scaled by the square root of the corresponding eigenvalue. As for PCA, the *N* first principal components are then taken as the *N* eigenvectors with the largest corresponding eigenvalue, with the length of the eigenvector scaled to match the square root of the corresponding eigenvalue.

With a given set of principal components vectors, less than the dimensionality of the data space, the data may be projected onto these vectors and thus be represented in the lower-dimensional principal-component coordinate system.

2.2.2 Elastic map

What the elastic map $[10, 11]$ does is providing a more generalized method of representing data from a high-dimensional space on a lower dimensional manifold. With a manifold given, any point from the data space may be projected onto the manifold, where the projected point may be represented both in the coordinate system of the original data space and the coordinate system of the manifold space. Then if the manifold encapsulates the general structure of the data well in the data space, the structure will be inherited in the manifold system as well, thus substantially simplifying any additional data analysis or interpretation.

The manifold itself, when expressed in the coordinate system of the data space, need not be linear. A linear manifold would be equivalent to a principal component analysis (PCA). However, a good low-dimensional representation of the data using PCA is dependent on the data being generally linearly correlated over the entire domain – an assumption that may prove to be erroneous. Elastic maps may here be regarded as a generalization, where the manifold is warped in the data space to better represent the data. This warping of the manifold is what may be denoted as the learning stage.

For some given manifold warped in the higher-dimensional dataspace one may define the *energy* of the manifold as the sum of the bending, stretching and approximation energy. The bending and stretching energy is a measure of the amount of warping and stretching of the manifold, where a minimization of the bending energy would result in a flat geometry of the manifold and a minimization of the stretching energy would result in a contraction of the manifold. Both of these are dependent only on the geometry of the manifold itself, regardless of any data points. In contrast, the approximation energy is a measure of the distance between the data points and the manifold. A complete minimization of this energy would result in the manifold passing through every data point in the data space.

Minimizing the summarized energy of this warped manifold is what is denoted as the *learning phase*. The mathematical description is given below.

Let $s \in S$ be data points in an N-dimensional Euclidean space (the data space), W be a set of nodes belonging to the manifold, J be the index set of the individual nodes of W (i.e. $w_j \in W \; \forall \; j \in J$), $K_j \subseteq S$ be the set of data points belonging to node w_j and $\|\cdot\|_{\mathcal{D}}$ be the Euclidean norm (i.e. the l^2 distance metric) in the data space.

The approximation energy is then defined as:

$$D = \frac{1}{2}\sum_{j\in J}\sum_{s\in K_j}\|s - w_j\|_{\mathcal{D}}^2$$

In the manifold space, two nodes that are directly adjacent form pairs connected by elastic edges and three adjacent nodes on a line form triplets connected by bending ribs. Let E denote the set of node pairs and G the set of triplets. The stretching and bending energy ($U_E,\ U_G$) on the manifold are then defined as:

$$U_E = \frac{1}{2}\lambda \sum_{(i,j)\in E}\|w_i - w_j\|_{\mathcal{D}}^2 \qquad U_G = \frac{1}{2}\mu \sum_{(i,j,l)\in G}\|w_i - 2w_j + w_l\|_{\mathcal{D}}^2$$

Here, λ and μ are two constants determining the overall weights for the elastic and bending energy.

The summarized energy, $U_{tot} = D + U_E + U_G$, is then to be minimized for given K_j. Due to the quadratic form of all terms, the problem is convex with respect to the nodal positions and the minimum energy can be obtained by solving for the nodal positions when the gradient of the summarized energy is zero. Taking the derivative of the summarized energy with respect to the nodal positions, a linear system of equations is obtained which, when solved for, gives the optimal solution for given K_j. Solving this system may be done using iterative or direct numerical methods, but since most nodes are only connected to their closest neighbors, the resulting linear system is sparse (i.e. mostly zero's) and direct numerical calculations are feasible.

With the energy minimized for a given clustering, K_j is updated for each node. Here, a data point is clustered to a node w_j if the distance between the data point and node w_j is the smallest for all nodes, i.e.

$$K_j = \left\{s : \|s - w_j\|_{\mathcal{D}} \leq \|s - w_l\|_{\mathcal{D}}\ \forall l \in J\right\}$$

With the new clusters K_j the minimum energy nodal positions w_j may be solved for again, after which new clusters K_j are calculated and so forth. This is repeated until the new clusters are equivalent to the old ones, in which case the algorithm is terminated.

Also, since the clustering choice function has the same form as the approximation energy function, the energy for any new cluster setup will always be lower than or equal to the old setup. Combined with the fact that the energy minimizing step given a cluster setup will also yield a solution with lower or equivalent energy due to the convexity property of the problem, the energy will always either decrease or remain constant between iterations. This guarantees convergence to at least a local minimum. Thus, this is a form of expectation-maximization algorithm.

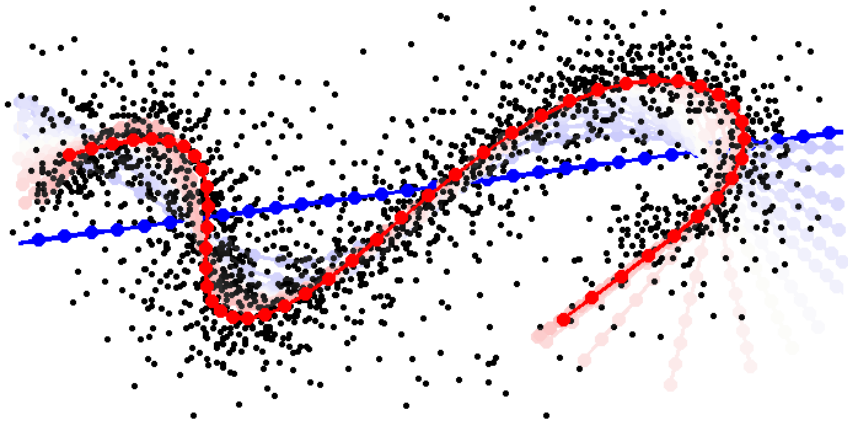


Figure 2.3: Example of 1-D elastic-map approximation of 2-D data

Figure 2.3 above depicts an example of how the EM-algorithm proceeds in fitting a 1-dimensional elastic map to a set of 2-dimensional data points generated for the purpose of visualization. The blue straight line represents the 1-dimensional PCA (which is used as the nodal starting positions). The red line represents the nodal positions after convergence.

2.2.3 Kohonen SOM

Self-organizing maps [12] is another dimensionality reduction method that produces similar solutions as elastic maps does, though the method of warping (i.e. *learning*) the lower-dimensional manifold is quite different. Instead of minimizing the overall energy of a manifold with respect to all the given data simultaneously, a single data point is fed to the algorithm at once after which the closest point in the manifold is found. The entire manifold is then dragged towards this data point, where the closest point in the manifold is dragged the most.

Let $\mathcal{D}$ and $\mathcal{M}$ denote the data space and manifold space respectively, $\|\cdot\|_{\mathcal{D}}$ and $\|\cdot\|_{\mathcal{M}}$ be the corresponding norms (i.e. distance function) to each space, $\lambda(n)$ the learning rate function and $\theta(u, v, n)$ the neighborhood function. Let S be the set of data points and W be the set of nodal points of the manifold.

The learning rate $\lambda(n)$ is set to be a monotonically decreasing function of the number of iterations n. It is set to be a value smaller than or equal to 1 in the first iterations and then to progressively decrease to 0 as the number of iterations increases. The value of this function represents how much a point in the manifold should be dragged towards a data point d; $\lambda = 1$ means that the manifold point is dragged all the way along a straight line towards the data point d and $\lambda = ½$ means that the manifold point is dragged half the distance along the same line.

The neighborhood function $\theta(u, v, n)$ is a monotonically decreasing function both of the number of iterations n and the distance between the two manifold points u and v as measured in the manifold space. This function determines how much any manifold point v should be dragged towards a data point d as its neighbor u is dragged towards the same data point, where u is the closest manifold point to the data point d as measured in the data space.

The properties of the learning rate and neighborhood functions are given below.

$\lambda(n) \in (0, a] \ \ \forall n \in \mathbb{N}, \ \ a \in (0,1]$
$\lambda(1) = a, \ \ \lim_{n\to\infty} \lambda(n) = 0,$
$\lambda(n+1) \leq \lambda(n)$

$\theta(u, v, n) \in (0,1] \ \ \forall u, v \in \mathcal{M}, \ \ n \in \mathbb{N}$
$\|u - v\|_{\mathcal{M}} = 0 \ \Rightarrow \ \theta(u, v, n) = 1 \ \ \forall n \in \mathbb{N}$
$\|u - v\|_{\mathcal{M}} \neq 0 \ \Rightarrow \ \theta(u, v, n) < 1 \ \ \forall n \in \mathbb{N}$
$\|u - v_1\|_{\mathcal{M}} < \|u - v_2\|_{\mathcal{M}} \ \Leftrightarrow \ \theta(u, v_1, n) < \theta(u, v_2, n)$
$\|u - v_1\|_{\mathcal{M}} = \|u - v_2\|_{\mathcal{M}} \ \Leftrightarrow \ \theta(u, v_1, n) = \theta(u, v_2, n)$
$\theta(v, v, n+1) \leq \theta(v, v, n)$

With suitable learning rate and neighborhood functions given the initial positions of the manifold points may be distributed randomly in the data space or by using some form of predefined mesh. Then any following algorithm iterations (sometimes referred to as *cycles*) are as follows (manifold point positions in the data space denoted as $v^{\mathcal{D}}$):

1. Choose a data point $s \in S$ at random

2. Find the manifold point $u \in W$ that is closest to data point s as measured in the data space, i.e.

$$u = \underset{v \in W}{\operatorname{argmin}}\left(\left\|s - v^{\mathcal{D}}\right\|_{\mathcal{D}}\right)$$

3. Update the data-space positions of all manifold points by 'dragging' them closer to s, weighted by the learning rate λ and the distance to u as measured in the manifold space, i.e.

$$\forall v \in W,$$

$$v_{n+1}^{\mathcal{D}} = v_n^{\mathcal{D}} + \lambda(n)\theta(u, v, n)(s - v_n^{\mathcal{D}})$$

2.2.4 Manifold output

With the nodes from the dimensionality reduction step given, the distance between these nodes and any point (i.e. vector) in the data space can easily be calculated using the standard Euclidean distance metric. With these distances given, the data vector may be expressed in the internal coordinate system of the manifold (e.g. by projection onto the manifold, or simply saying that the vector belongs to the nearest node). In this setting, the data vector is dimensionally reduced to a single point in the two-dimensional manifold space. If successful, the different classes/behaviors under consideration should be placed separately on the manifold. Cluster analysis may thus be appropriate for this setting, given that the classes are fairly well separated.

It is, however, not the only option. Instead of projecting the vector onto the manifold, where the vector is dimensionally reduced to a single point in the two-dimensional manifold, each node in the manifold may be given an output based on the distance between each node and the vector (as measured in the data space). Preferably, this output should be inversely proportional to the distance, meaning that the further the vector is from the node, the smaller

is the output from that node. This output for each node may, for example, be based on a Gaussian function or any other monotonically d**ecr**easing function. The output may further be normalized with respect to the summarized output from all nodes or to make it fit some predefined range. In either case, the output generated in this setting will not be a single point but a distance-dependent imprint on the entire manifold. This imprint will henceforth be denoted as a *signature*.

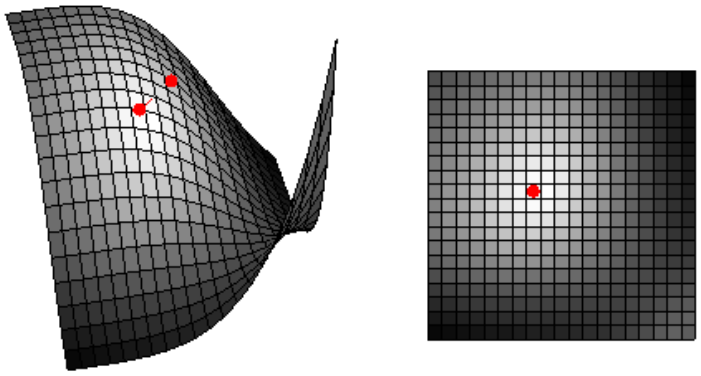


Figure 2.4: Example of manifold output
(signature inversely proportional to distance)

Using a Gaussian for calculating outputs for each node has some advantages. One is that the output from each node will always remain bounded between 0 and 1 regardless of normalization procedure (range of manifold output or summarized output), another is that the *signature*-based output type is reduced to normal projection when the parameter describing the amount of exponential decrease is set to 0 – that is, the projection is a special type of the *signature*-type output. Specifically, defining the output from each node by

$$O_n = e^{-\left(\frac{\|s-v_n^{\mathcal{D}}\|}{B}\right)^2}$$

where s is any CDF-vector (i.e. data point) fed to the system, $v_n^{\mathcal{D}}$ is the nodal position of node n in the data space, B is the parameter describing the amount of exponential decrease per unit length and O_n is the output from node n, it is quite clear that the largest output will tend to dominate. To show this even more clearly, divide all outputs by the maximum output. Then the new maximum output will be equal to 1 and all other outputs strictly smaller than 1, assuming the maximum output is unique. Then, for any other output,

$$O_n = e^{\frac{d_{min}^2 - \|s-v_n^{\mathcal{D}}\|^2}{B^2}}$$

where

$$d_{min}^2 = \min_n \left(\|s - v_n^{\mathcal{D}}\|^2\right)$$

we have that for any other node but the nearest node,

$$d_{min}^2 - \left\| s - v_n^{\mathcal{D}} \right\|^2 < 0 \quad \Rightarrow \quad \lim_{B \to 0^+} \left(\frac{d_{min}^2 - \left\| s - v_n^{\mathcal{D}} \right\|^2}{B^2} \right) = -\infty \quad \Rightarrow \quad \lim_{B \to 0^+} \left(e^{\frac{d_{min}^2 - \left\| s - v_n^{\mathcal{D}} \right\|^2}{B^2}} \right) = 0$$

3 Data structure and methodology

3.1 Given type of data

The data is given in the form of time series where 10 target properties are updated roughly once every second. From these properties an additional set of 6 properties may be inferred that are much more easily understood, namely the positions in Euclidean coordinates and the velocity components for each Euclidean direction. The following tables contain the given and calculated properties.

Given

Name	Unit	Description
t	(s)	Time
Be	(^{o})	Bearing
Di	(m)	Distance to radar
El	(^{o})	Elevation
C	(^{o})	Course
V_{tot}	(m/s)	Absolute velocity
RCS	(m^2)	Radar Cross Section
V_{Di}	(m/s)	Radial velocity
V_{Be}	(^{o} /s)	Bearing differential
V_{El}	(^{o} /s)	Elevation differential

Calculated

Name	Unit	Description
x	(m)	x-position
y	(m)	y-position
z	(m)	z-position (Height)
V_x	(m/s)	Velocity in x-direction
V_y	(m/s)	Velocity in y-direction
V_z	(m/s)	Velocity in z-direction

The total set of data points for each specific target is denoted as one *track*. The length of a track can vary greatly, typically depending on whether the target remains within range of the radar and if the radar software can properly distinguish the target data from background noise. The latter imposes an additional problem when the target is on the verge of being detected – the time series data for the track contain "holes". It skips some updates. This problem is handled by preprocessing the data and filling these holes using some suitable interpolation technique (simple linear interpolation was used in this case).

3.2 The problem with distance measures

The main purpose of this project is to be able to classify specific behaviors, pre-specified or anomalous, based on a given time series of data. This objective will inevitably lead to the necessity of being able to measure distances between different time series, which is a truly non-trivial matter.

Treating the data as points embedded in a high-dimensional Euclidean space, equipped with the standard Euclidean distance measure, poses several problems. One of the most obvious problems is the necessity of the time series having the same length. Another is a possible lack of autocorrelations in the time series, meaning that there is a low correlation for the time series with itself if it is shifted in time. However, the most serious problems arising when attempting to deal with the data in this manner is commonly denoted as the *curse of dimensionality*[13]. This is not a single problem but a set of problems encountered when attempting to classify high-dimensional data – problems that are usually not seen when working in low-dimensional spaces.

The problem specific for the choice of a distance function, in particular the Euclidean distance, may be illustrated in several ways. One way is to compare the volume of a hypersphere embedded in a hypercube of the same dimension to the volume of the hypercube itself. Doing so one finds that the volume of the hypercube grows much faster than that of the hypersphere as the dimensionality increases and thus that the ratio between the volume of the sphere and the cube tends towards zero, meaning that under the assumption of uniformly distributed data, most points will be found in the corners of the hypercube.

Another way to illustrate the deteriorating notion of *distance* for these types of distance measures is to first assume that the data points are placed randomly and given a coordinate for each dimension governed by some distribution *R*. Then, by the definition of the Euclidean distance function, the measured distance between some reference point *P* and a data point placed in this manner will in effect be the sum of *d* independent and identically distributed (IID) random variables, where *d* is the number of dimensions and the distribution under consideration, which may be denoted as $\tilde{R}_j$, is the squared distance between the reference point *P* and the distribution *R* in dimension *j*.

By the *law of large numbers*, the sum $\frac{1}{d}\sum \tilde{R}_j$ will tend towards the mean of $\tilde{R}$ with a diminishing standard deviation. What this implies is that any finite set of points placed randomly in this manner will in effect all have the same measured distance to the reference point *P*.

One way to resolve these problems is to assume that the *distribution* for the target properties is sufficient to classify the behaviors. In other words, if each new entry in the time series is treated as a random variable *X*, it is assumed that the distribution of *X* is different for different behaviors. In practice, a distribution for the properties is generated for each track and this distribution is compared to a stored set of distributions of known classification to find a best match. This type of convergence is commonly called *convergence in distribution* [14].

If it is assumed that each distribution belongs to a specific type of distributions (e.g. Gaussian, exponential etc.) it is possible to represent each distribution using a small set of parameters (e.g. mean, variance, skewness etc.). It is also possible to make no assumption at all of the underlying type of distribution and make a comparison of the of the measured distribution data directly. The latter approach will be employed in this thesis, in practice by taking a finite set of equally spaced points along the estimated distribution.

An additional intriguing feature of comparing distributions is that one also relaxes the constraint of the time series having the same length.

3.3 PDF/CDF

When talking about comparing distributions it is not stated whether it refers to the probability density function (henceforth denoted as *PDF*) or the cumulative distribution function (henceforth denoted as *CDF*). Both of these could work as a choice of distributions to work with since convergence in any of these, in this setting, implies convergence in the other. However, there are different properties for the PDF and CDF that could be difficult to deal with for the proposed classification methods and, as such, the choice of type of distribution must be specified beforehand. Of course, this choice is irrelevant if the distribution function is pre-specified; If it assumed to be Gaussian, the calculated parameters do not depend on whether the distribution is given in the form of a PDF or CDF. This choice only concerns the case when no assumption of underlying distribution function is made.

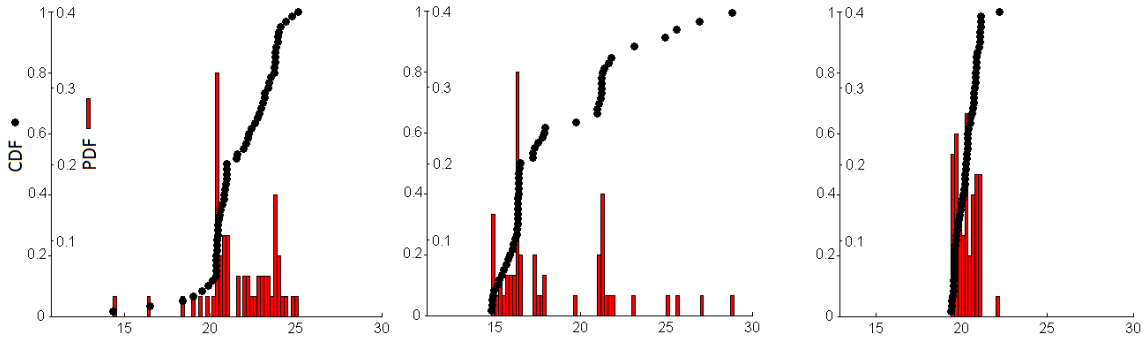


Figure 3.1: Example of PDF and CDF of the same property for three different cases

In practice, for the case of the PDF, the distribution may be represented using a finite set of points by taking the probability density at different positions along the x-axis. This can be done for the CDF as well. For the CDF, however, there is an additional method. For any probability on the CDF-curve (i.e. any position on the y-axis) there is only one corresponding position on the x-axis. This means that the CDF curve is *invertible*, that is, there exists an *inverse*. This second method for the CDF is thus to take points along the y-axis and find the corresponding x-value.

Different problems are imposed by these methods. One common problem is encountered for the method of taking values along the x-axis, both for the PDF and CDF distributions. This problem is due to the distributions being heavily shifted along the x-axis for different classes. Being able to encapsulate the distributions properly would require sampling over a wide range along the x-axis, which is equivalent to representing the distributions using a large set of values. This is especially true for the PDF case since there are narrow sections where the probability density is big, thus requiring the sampling positions being densely distributed along the x-axis. Also, if the set of data points is too small, using the PDF would additionally require some method of density estimation since a normal histogram would likely be insufficient.

One problem common for all methods, but especially for the CDF, is the correlation between subsequent sampling points. If both the PDF and CDF are continuous functions (which they are assumed to be here, at least in the limiting case of an infinite amount of data), points that are close to each other along the x-axis will also be close along the y-axis. This correlation is even larger for the CDF since it is a monotonically increasing function; a larger x-value always corresponds to a larger or equal y-value. The hope is that this correlation will be accounted for in the dimensionality reduction step.

The type of distribution chosen is the CDF, inversely sampled. The reason is the relatively small amount of sampling points needed as well as the non-necessity of using any type of density estimation.

The number of measurement points placed in the CDF is set to 60, corresponding to one whole minute of measurements of a target. This is roughly the longest period that can be accepted for a target to remain unclassified, and for the sake of the generated distributions, the longer the period the better. All tracks are therefore split into slightly overlapping 60-second intervals, each treated as an individual measurement. It is, from each of these intervals, that a distribution of some statistical value will be calculated.

3.4 Methodology

The overall method that will be employed in this thesis is to first derive a set of relevant statistical values, empirically or theoretically justified, and then to dimensionally reduce the space of statistical values to a Euclidean 2-dimensional space. Cluster analysis will later be conducted in the dimensionally reduced space. The figure below gives a schematic view of this process. In this case, there are two main behaviors to be classified, represented as red and blue.

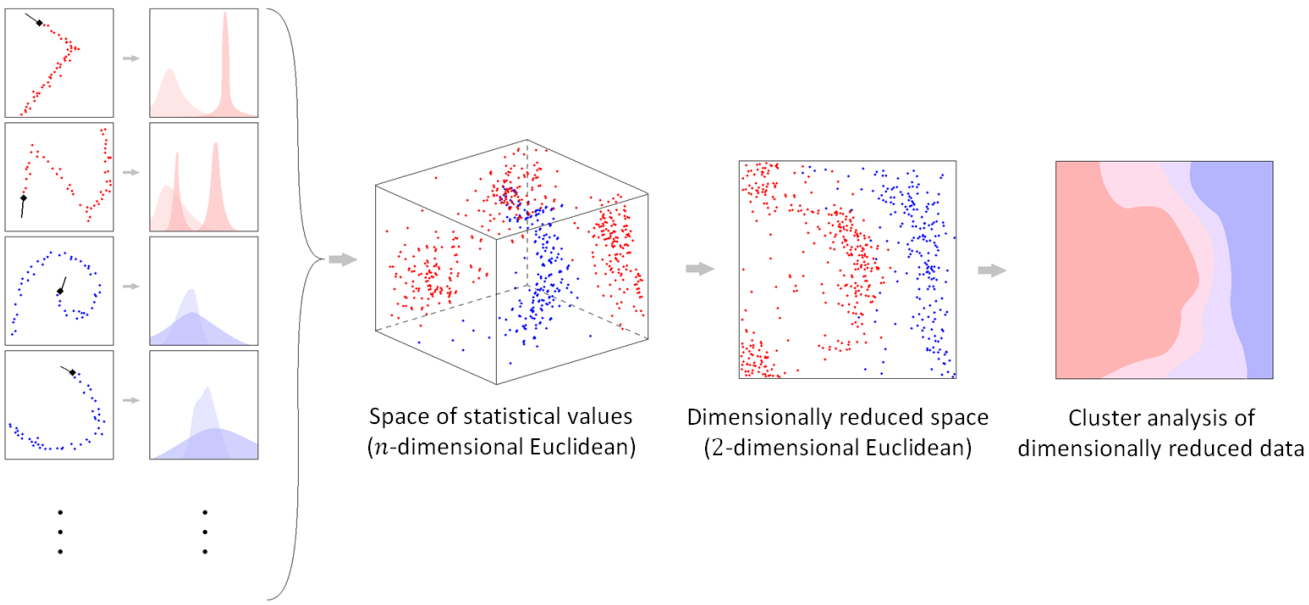


Figure 3.2: Schematic view of classification procedure

3.5 Chosen statistical values

For this project, 8 different statistics have been chosen to be used in the classification procedure. The choice of these statistical values is only vaguely theoretically justified – rather it is a highly arbitrary choice of values that might, naively reasoned, correspond to the different classes to be identified. Some values are also chosen based on some apparent visually identified differences in the cumulative distributions for the different classes.

Before all statistical values are presented it should be noted that the Euclidean coordinates, for some of these statistical values, are processed further still, namely by applying a smoothing spline on the positions. The reason is some seemingly erroneous positional updates for some tracks. These spline-smoothed coordinated will be denoted by a small *spline*-notion. Spline-smoothed values for the velocities are found by calculating the distance between succeeding spline-smoothed coordinates.

The following list contains all statistical values under consideration.

$\text{sign}\left(\text{mean}\left(\Delta C^{spline}\right)\right) \cdot \Delta C^{spline} = S_1$: Course change according to the spline-interpolated coordinates

$\log(|z - \text{mean}(z)|) = S_2$: Difference between current altitude and mean

$\log(|V_z - \text{mean}(V_z)|) = S_3$: Difference between current altitude velocity and mean

$\log(|\Delta C|) = S_4$: Absolute value of course change (without applying smoothing spline)

$\log(RCS) = S_5$: Radar Cross Section

$\log(V_{tot}) = S_6$: Total velocity

$\log\left(\left|\frac{RCS_t}{RCS_{t-1}} - 1\right|\right) = S_7$: Relative difference between succeeding updates for RCS

$\log\left(\left|\frac{RCS}{V_{tot}^{spline} - V_{tot}}\right|\right) = S_8$: Current RCS divided by difference between smoothed and non-smoothed velocity

$\boldsymbol{S_1, S_4}$: ΔC is the course change between subsequent measurement updates as calculated for each 60-second interval. The idea behind S_1 is to properly distinguish a circulation behavior from more or less straight movements. The *sign*-function for S_1 is applied in order to make all circulating behaviors clockwise. The idea behind S_4 is to distinguish erratic behaviors from smooth ones. It does not depend on which way the target turns, but on *how much* it turns and how often this occurs.

$\boldsymbol{S_2, S_3, S_6}$: These values are motivated by the idea that the classes to be properly distinguished will either retain the same altitude, make changes of altitude at roughly the same

pace throughout or keep some specific velocity more or less constant.

$\boldsymbol{S_5, S_7}$: The motivation behind these values is both the absolute of the apparent surface area of the target and the measured differences between subsequent updates.

$\boldsymbol{S_8}$: There is no real theoretical justification for using this value other than some apparent visual differences in the plotted cumulative distributions.

3.6 Sampling and concatenation

It should hereby be stressed that the distribution for *several* quantities will be used simultaneously in the classification procedure, not just a single one. As such, the sampled points from each distribution must be added together in some form and treated as a single vector of sampled points. This is done by simple concatenation; The sampled points for the first distribution are placed on the first rows in this sample-vector. Then, the sampled values from each succeeding distribution are placed 'below' the former ones, thus forming the sample-vector. The figure below gives a graphical explanation of this procedure.

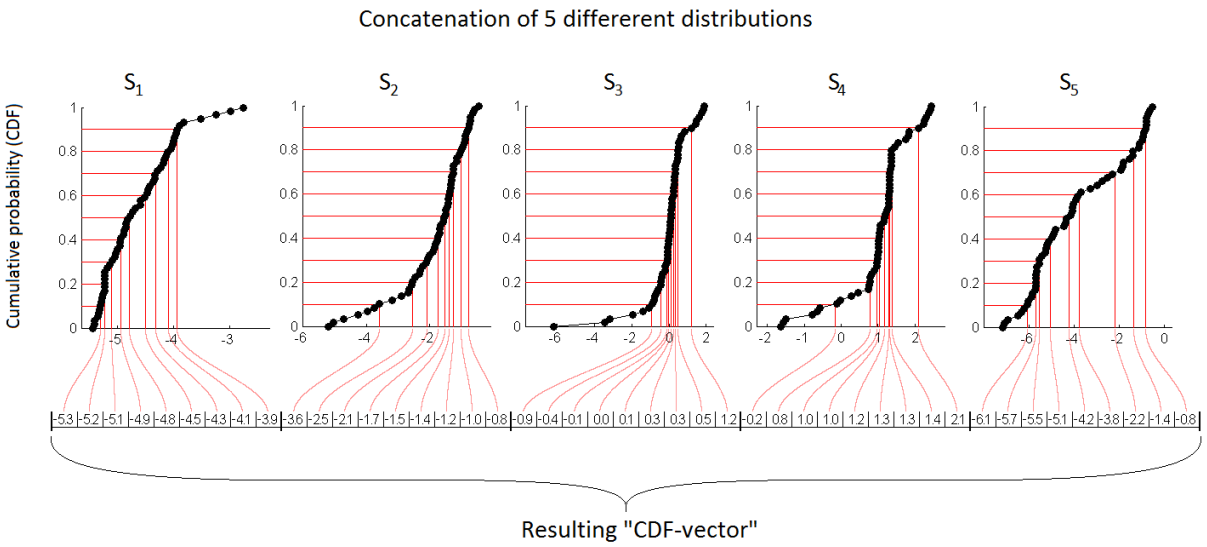


Figure 3.3: Concatenation of distributions

The number of points along the y-axis (i.e. the number of sampling points for each distribution) may be chosen arbitrarily. In this case, 15 equally distanced sampling points are placed along the y-axis. In order to avoid problems related to outliers, the entire range is not considered. Instead, the sampling points are placed between 0.05 and 0.95 (i.e. between the 5'th and 95'th percentile). If any sampling point does not correspond perfectly to a measured value placed in the empirical CDF, the value is estimated using linear interpolation.

With 8 distributions and 15 sampling points from each the resulting CDF-vector will contain 120 entries. The choice of 15 sampling points is quite an arbitrary choice and merely motivated as a suitable number to fulfill the criterion of maximum amount of required computational time. See the next chapter on *computational complexity*.

4 Regarding computational complexity

Some of the computational complexity encountered during system training and operation (which are considered separate) are examined in this section. It is henceforth in all descriptions of the manifold assumed that the manifold itself consists of a regular 2-dimensional grid.

4.1 Operation

System operation refers to the real-time functioning system operating with a given set of fixed parameters. Analyzing the computational complexity is thus the evaluation of the computational time required to generate and sample from the chosen statistics, dimensionally reduce the generated CDF-vector using the manifold nodes and then to classify the resulting dimensionally reduced vector using some form of cluster analysis.

4.1.1 Generating CDF and sampling

Generating the CDF will typically involve calculating the desired statistical values and sorting them ascendingly. The computational time will thus at least be proportional to the product of the length of the time series and the number of different statistical values under consideration. In addition, a sorting algorithm is to be applied to the resulting list of values after which interpolated points in this list are to be calculated.

If we by n denote the number of points in the generated distribution, the computational time required for sorting will in general be proportional to $n\log(n)$, assuming an efficient sorting algorithm is used$[15]$. As for the interpolation step it will require both the sorting of sample-data-list and the operations to estimate the values at each sample point. If we by m denote the number of sample points, the sorting of the list will thus be proportional to $(m+n)\log(m+n)$ while the sample value estimation will simply be directly proportional to m.

As such, the overall computational complexity for these steps is proportional to $(m+n)\log(m+n)$, where m and n are the number of sample and data points respectively

4.1.2 Dimensionality reduction

Assuming a manifold consisting of N nodes embedded in a d-dimensional (Euclidean) data space, dimensionally reducing the d-dimensional CDF-vector will involve calculating the distance to the manifold, thus making N distance calculations. Using the Euclidean metric, each distance calculations will be the summarized squared distance in each dimension - a sum of d different terms. As such, the computational time required for calculating the distance from a data point to the manifold will be proportional to $d \cdot N$. Here, d is the total number of sampling points summarized for all distributions.

However, the computational time will in general be independent of the desired type of manifold output as all output types requires this manifold distance calculation. For example, the projection type requires finding the nearest node to the data point which leads to the necessity of measuring the distance between the data point and *all* nodes in the manifold. Using the signature output method also requires this calculation, but here every manifold node is given an output based on this distance. Thus, the computational complexity with respect to the number of nodes will be the same for both of these output types.

4.1.3 Classification

Depending on the type of classification method to be used computational time for this part may be already incorporated in the previous steps. Treating each manifold node as a member of either class, the k-NN algorithm using the data-space distances can be directly applied to the distances calculated in the dimensionality reduction step (in fact, dimensionally reducing the CDF-vector to a point on the manifold will then be unnecessary). If, however, the distances are calculated as internal distances on the manifold, the distances between the dimensionally reduced data point and the manifold nodes must then be calculated. This corresponds to $N \cdot d_{\mathcal{M}}$, where N is the number of nodes and $d_{\mathcal{M}}$ is the dimensionality of the manifold.

As for the *signature*-type classification, each signature will consist of N entries which are to be compared to a set of stored signatures of known class to find the best correspondence. Using the Euclidean metric for these calculations, the computational time required will thus be proportional to $N \cdot n_{signatures}$. The number of typical signatures stored need not be the same as the number of classes to be identified, as several signatures may belong to the same class. However, every class must contain at least one signature.

It has been previously specified that each time series under consideration is to be 60 entries (i.e. seconds) long, where the time series are used in the creation of 8 different statistical values. The distributions for these values are then sampled using 15 sampling points for each distribution, resulting in a total of 120 entries in the CDF-vector for each time series.

The requirement of being able to classify 1000 targets each second poses no problem during generation of the CDF's. However, the dimensionality reduction and classification step may need some carefully chosen setups as to avoid overloading.

A manifold consisting of 30×30 nodes (900 nodes in total) combined with the usage of at most 120 typical signatures is on the verge of clearing the 1-second mark for CDF generation, dimensionality reduction and classification. This will henceforth be the setup used.

4.2 System training

System training refers to the process of finding all system parameters based on a given set of data. Specifically, it incorporates the process of finding the manifold nodes for the dimensionality reduction part and possibly the case of cluster analysis, e.g. calculating manifold areas that belong to specific classes or inferring a set of typical signatures for each class.

For both the elastic map and Kohonen SOM, all the given data must be clustered to its nearest node (in each iteration for the elastic map and in each 'cycle' for Kohonen SOM). Using the Euclidean metric, the distance between a data point and a node will be summarized squared distance in each dimension, thus scaling this computational time linearly with the number of dimensions d. Combined with the requirement of calculating the distance between every data-node-pair, the total computational time for clustering the data to its nearest node will scale according to

$$t_{comp} \propto d \cdot N_{node} \cdot N_{data}$$

4.2.1 Elastic map

Here, all grid nodes are connected in a regular mesh-like structure where two neighboring nodes are connected forming "elastic edges" and three adjacent nodes are connected forming "bending ribs". If the manifold is 2-dimensional, any internal node (i.e. not at the edge of the manifold membrane) will be part of four elastic edges and six bending ribs. In total, each node will be connected to eight other nodes. This implies that each row in the system of linear equations resulting when minimizing the quadratic summarized energy of the manifold will contain at most 9 non-zero elements, one of which will always be at the diagonal. In addition, if the enumeration of the nodes in the manifold follows an efficient system (e.g. the enumeration moves along one "line" in the manifold grid at a time) the resulting matrix will also be banded (all non-zero elements are confined to a diagonal band centered on the diagonal).

As such, in each expectation step in the EM-algorithm used to find the nodes for the elastic map system, a sparse, banded matrix system is to be solved, where the size of the matrix system corresponds to the total number of nodes in the manifold. It is a very good idea to take advantage of the sparse structure of this matrix when solving the system. In Matlab, the user may specify the structure of the matrix (i.e. using the *sparse*-command) as well as the desired method of solution (i.e. direct or iterative methods). Matlab uses direct numerical methods unless an iterative method is specified.

Figure 4.1 below depicts the advantage of using the built-in sparse system solver. It is a log-log-plot of the number of rows in a sparse matrix (generated in the same way as the systems encountered in the elastic-map algorithm) versus the computational time as measured using Matlab's timer function. The inclination of the resulting scatterplots gives a hint of the polynomial order of the computational time. It is here seen that the computational times increases almost directly proportional to the number of nodes when using the *sparse*-solver, while it increases almost cubically when using the standard full-size form. However, this difference is only seen when using more than a few hundred nodes.

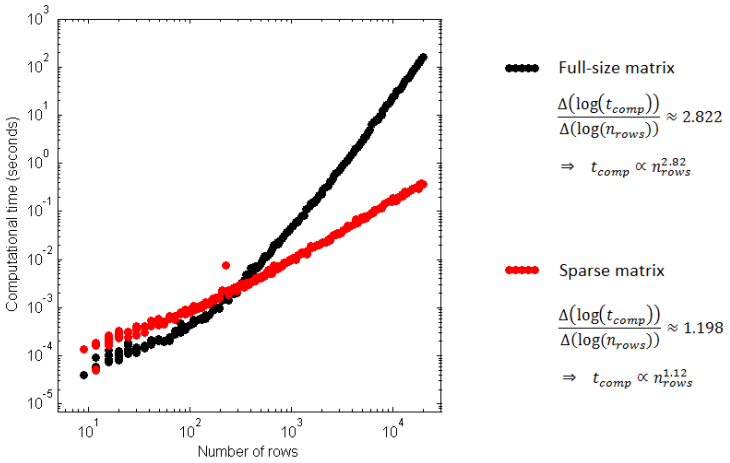


Figure 4.1: Measured computational time for matrix solution when using full-size and sparse matrices

4.2.2 Kohonen SOM

For the Kohonen SOM, in each iteration a single data point will be fed into the system and the manifold nodes will be dragged towards this data point. Feeding the system with all the data points is denoted as one *cycle*. The computational time for one *cycle* is considered henceforth.

First, in each iteration, the nearest manifold node to the data point is to be found. Using the Euclidean metric this computational time will be equivalent to that of the clustering part described for the elastic map method. Second, each node in the manifold is to be dragged towards this data point where the amount of "dragging" is determined partly by the learning rate but, most importantly, by the distance between each node and the nearest node *as measured in the manifold space*. This is commonly denoted as the neighborhood function. What this implies is that the neighborhood function value between two nodes is independent of the manifold warping in the data space. Thus, it is possible to calculate all the internal distances in the manifold beforehand and then use these distances when calculating the values of the neighborhood function. This will save a whole lot of computational time, especially if the number of nodes is large.

In summary, both the elastic map and Kohonen SOM will scale approximately according to the product of the number of nodes, number of data points and dimensionality of the data space.

5 Experimental setup and result

5.1 System setup

The following list describes the system setup for the elastic map and Kohonen SOM algorithms. For the elastic map, two parameters are set (informally representing the contractive strength and rigidity of the resulting manifold, respectively) and for Kohonen SOM, the neighborhood and learning rate functions are defined as well as corresponding parameters to these functions.

Elastic map

Manifold nodes	$900\ (30 \times 30)$
Nodal starting positions	2-D PCA
Elasticity	1
Rigidity	10

Kohonen SOM

Manifold nodes	$900\ (30 \times 30)$
Nodal starting positions	2-D PCA
Learning rate function	$\lambda(n) = \lambda_0 \mathrm{e}^{-n/\alpha}$
Neighborhood function	$\theta(u, v, n) = \mathrm{e}^{-\left(\frac{\|u-v\|}{\beta_0+\beta_1 \cdot n}\right)^2}$

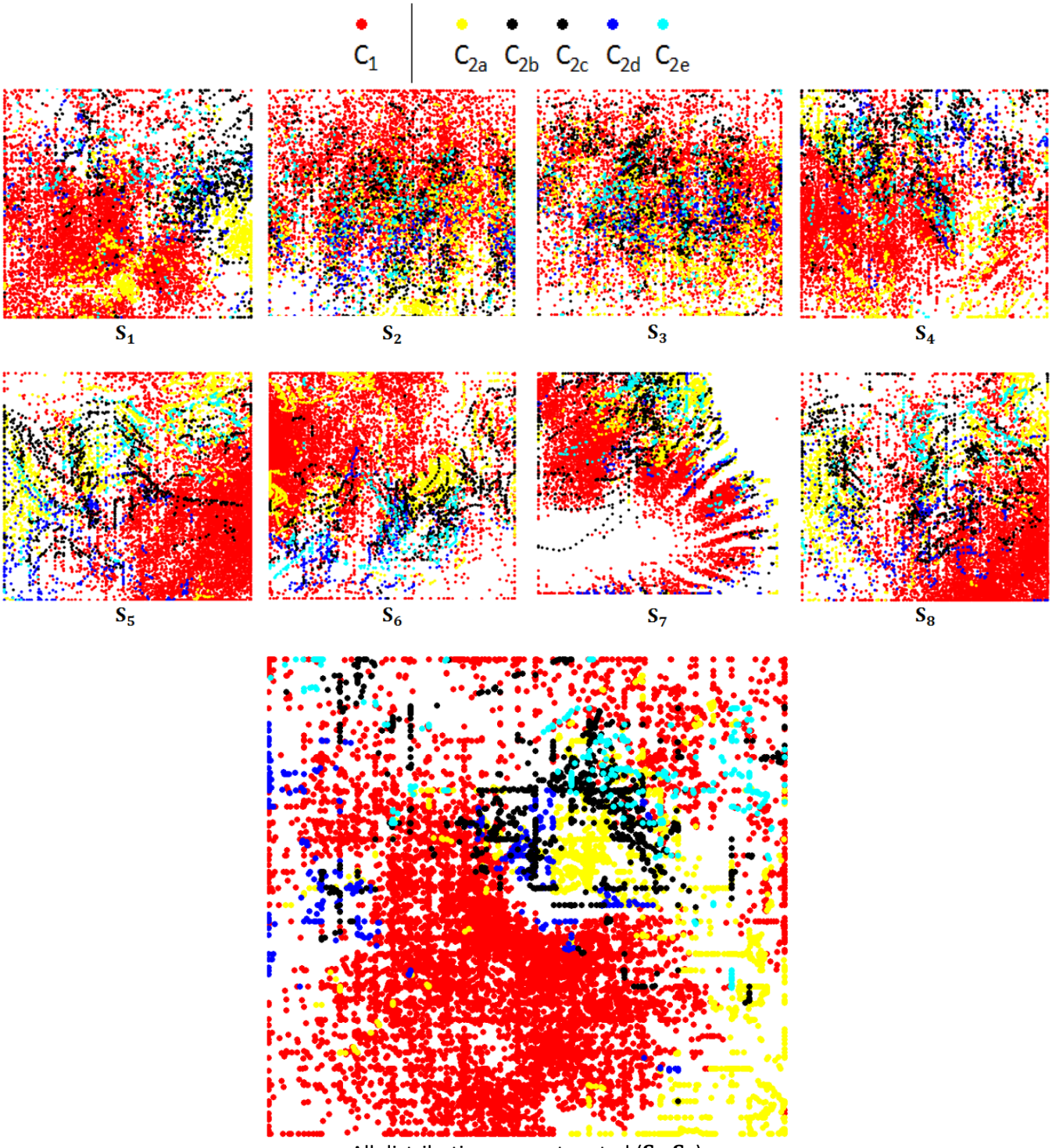
$$\begin{aligned} \lambda_0 &= 0.5 \\ \alpha &= 4 \cdot 10^3 \\ \beta_0 &= 4.6 \\ \beta_1 &= -4.5 \cdot 10^{-5} \end{aligned}$$

All parameters, both for the elastic map and for Kohonen SOM, are set somewhat arbitrarily to values that seemed to result in some fairly good clustering when tested on a small amount of data. The resulting manifold projections when using these parameters are shown in the following section.

5.2 Resulting manifolds

5.2.1 Elastic map

The figures below show the resulting data projections on the manifold after the manifold have been warped using the elastic map methodology. The eight different distributions have all been used to warp the manifold separately creating eight different manifold setups, as well as combined, creating the manifold shown as the largest subfigure. As can be seen, the combination of all distributions gives the greatest separation of the classes.



All distributions concatenated ($\mathbf{S_1}$-$\mathbf{S_8}$)

Figure 5.1: Projection on elastic-map-manifold

5.2.2 Kohonen SOM

The figures below show the resulting data projections on the manifold after the manifold nodal positions have been found using Kohonen SOM. It is here evident that the result is fairly similar to that obtained when using the elastic map methodology, both in the sense of the similar projections and that the best separation of the classes is obtained when using all distributions simultaneously.

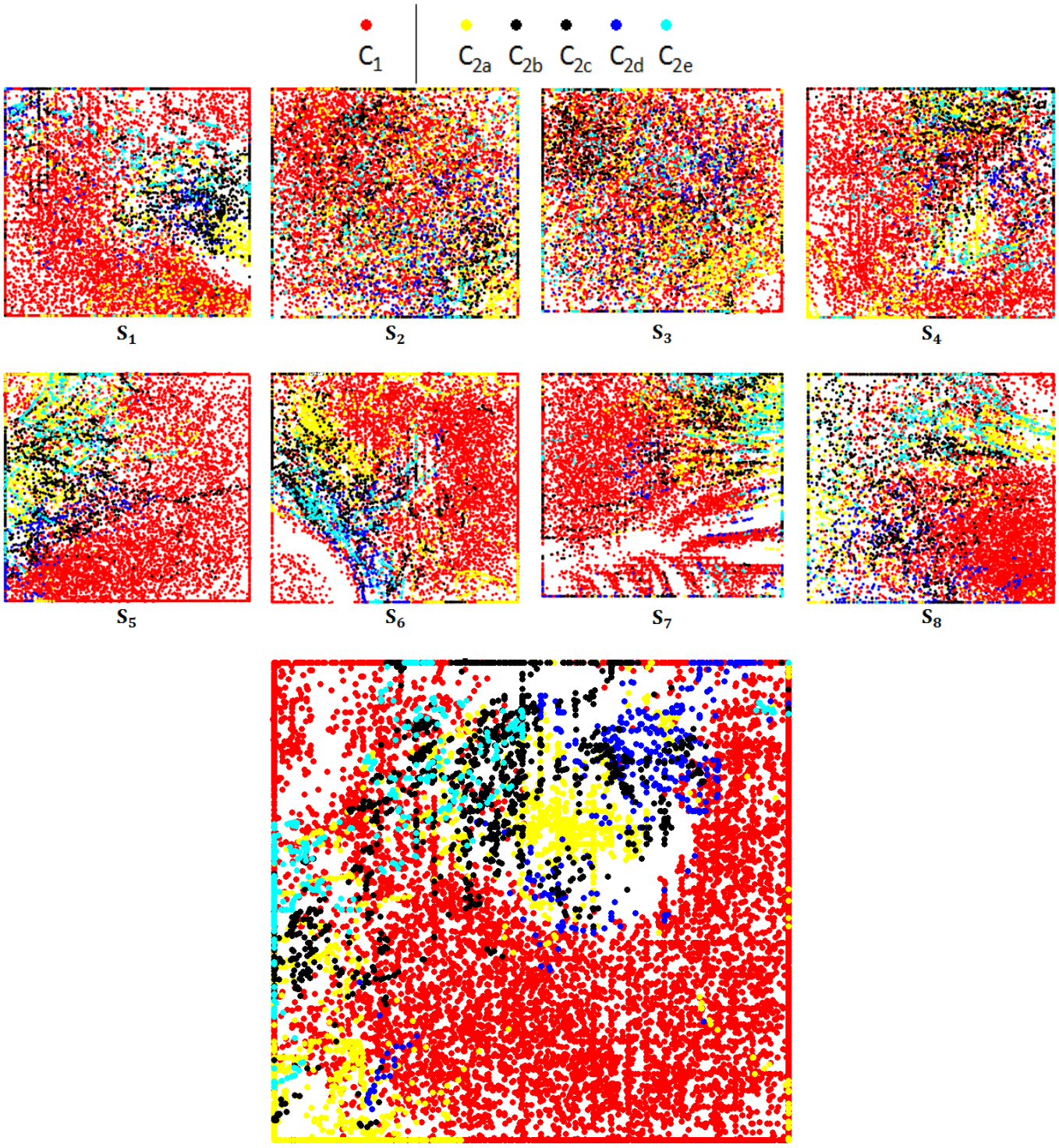


Figure 5.2: Projection on Kohonen SOM-manifold

5.3 Classification

The table below contains the classification result when using the standard setup for the elastic map and Kohonen SOM methods.

Correct classification (%)	Training (Validation)	
	C_1	C_2
Elastic map	86.4 (86.5)	91.7 (80.3)
Kohonen SOM	83.8 (83.6)	88.7 (78.3)

The following figures show some examples of the classification correspondence for C_1 and C_2. In all eight figures, a 60-second CDF is generated based on the time-series past 60 updates. The four left figures are examples for known classes of C_1 and the rights are for C_2. In all eight figures, the red line corresponds to the classification value for C_1 and the black line for C_2. The top line will yield the classification. The *value* on the y-axis is of less importance; It corresponds to the (summarized) inverse distance to the class-clusters.

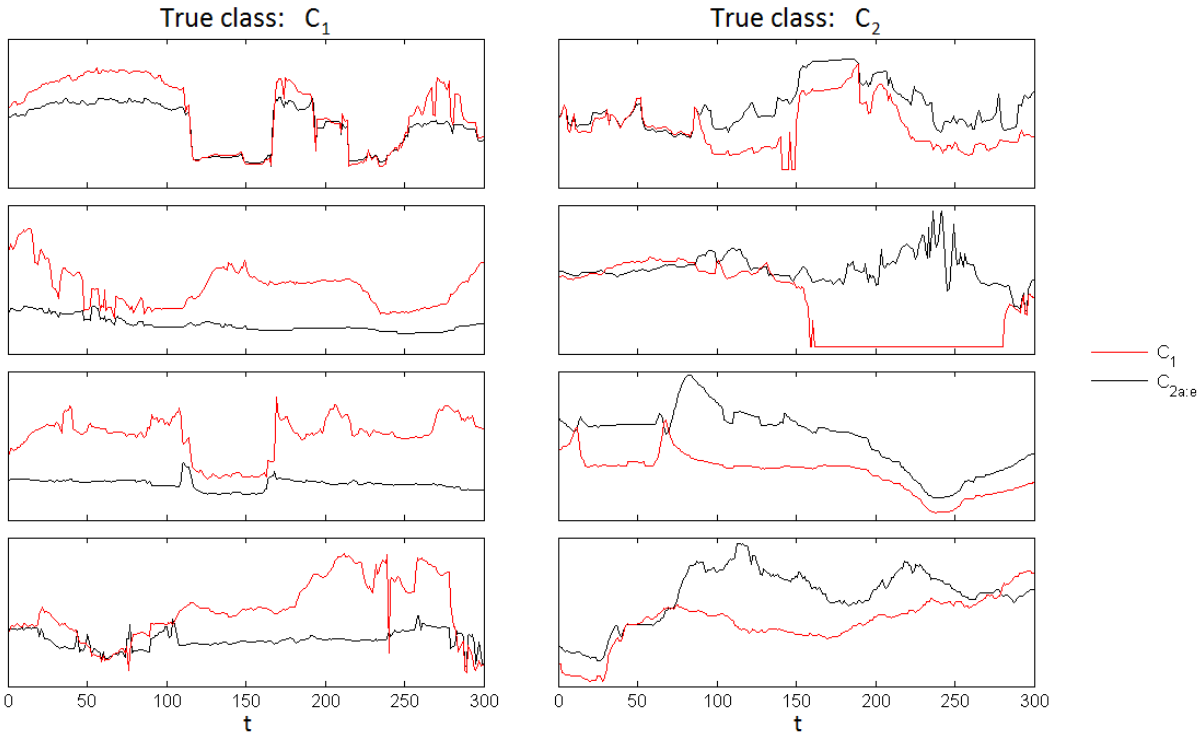


Figure 5.3: Classification for 8 examples of (at least) 360-second long time series for class C_1(left) and C_2(right). At each time step, the CDF is generated using the past 60 time series updates.

5.4 Changing model parameters

In this section the effect of changing the model parameters is investigated. The parameters under consideration are those that are common for both the elastic map and Kohonen SOM methods. This incorporates the time series length considered and all parameters related to the manifold output and clustering. The manifold nodal positions used for all these cases were found using the elastic map algorithm.

Using a Gaussian for calculating the manifold output combined with a k-NN classification, the following figures depict the resulting classification errors for different sets of system parameters. As the "main" set, the exponential-decay-parameter 'B' is set to 10, the total number of typical signatures to be found is set to 50 ($[C_1, C_{2a}, C_{2b}, C_{2c}, C_{2d}, C_{2e}] = [25, 6, 6, 1, 6, 6]$), the number of neighbors considered in the k-NN classification step is set to 10 and the weighting for each neighbor is set to be inversely proportional to distance.

5.4.1 Exponential-decay parameter B

The following figure depicts the effect of changing the exponential-decay-parameter B while keeping all other system parameters constant. The clustering was conducted 20 times for each B-value. Each time a clustering was conducted, different clustering means was found (due to the randomized initiation of the cluster means in the EM-algorithm) and, as such, different classification results was obtained each time the classification was conducted. The bars show the percentage of correct classification for C_1 (red) and the mean of all subclasses in C_2 (black) in the form of 25% - 75% quantiles. The subfigure shows Pearson's correlation coefficient. A negative value of the correlation coefficient implies that if the correct classification percentage was high for C_1, it was low for C_2 and vice versa.

A smoothing spline has been applied to the empirical values of the mean and upper/lower quantiles to better visually emphasize the general changes in the classification results.

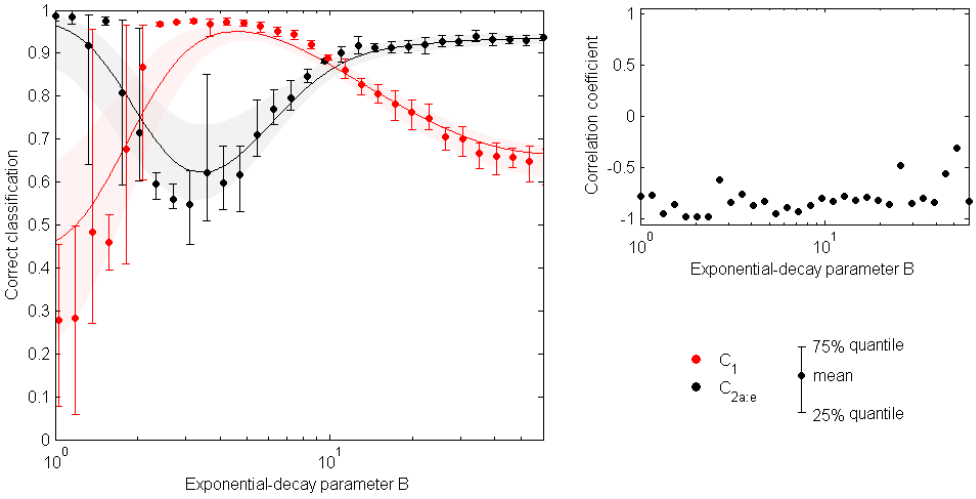


Figure 5.4: Classification result when changing the exponential-decay parameter B

5.4.2 k-NN neighbors

The following figure depicts the effect of changing the number of neighbors considered in the k-NN classification step. The clustering was conducted 15 times for each k-NN value between 1 and 20 and due to the randomized initiation in the EM-algorithm, 15 different classification results were obtained for each k-NN value. The mean and band between lower/upper 25% - 75% quantiles are shown for C_1 and C_2. The subfigure shows Pearson's correlation coefficient.

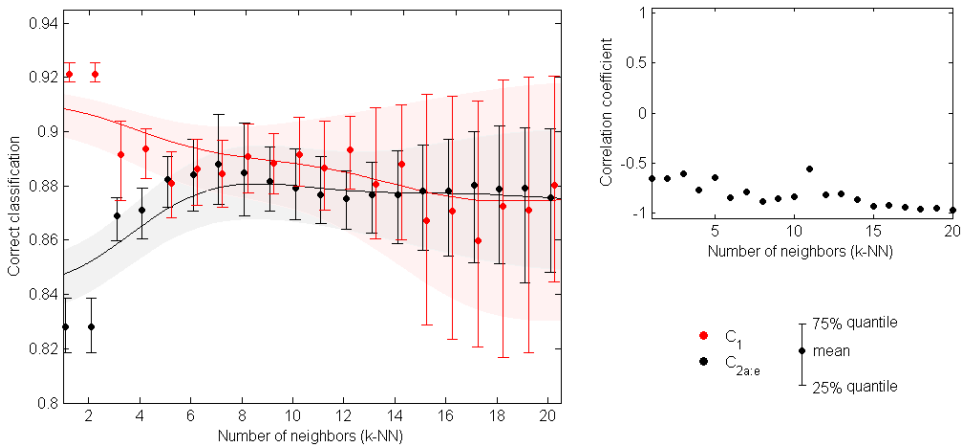


Figure 5.5: Classification result when changing the number of neighbors considered in the k-NN classification algorithm

5.4.3 Time series length

The following figure shows the effect of using different time-series lengths. Note that different time series lengths for classification and manifold estimation are used here. There are 5 subfigures where in each subfigure a different time-series length for the manifold generation is used (set to $30, 45, 60, 90$ and 120 seconds) and for each manifold, classification-CDF's varying between 20 and 120 seconds are dimensionally reduced and clustered on these manifolds.

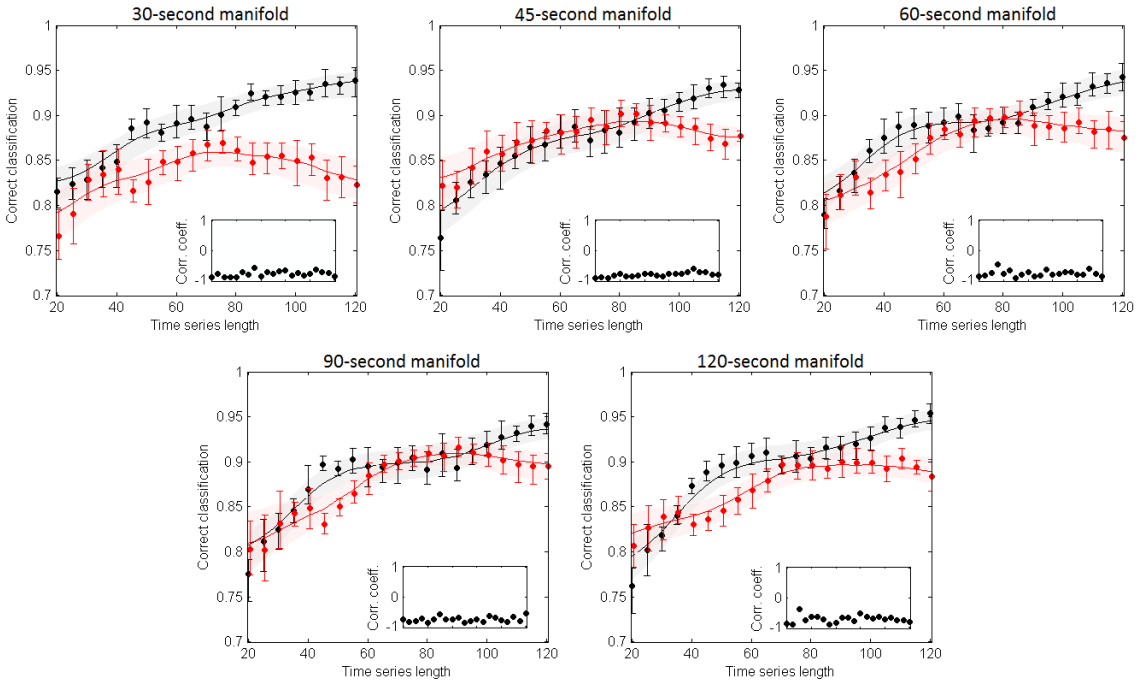


Figure 5.6: Classification result when changing the length of the time series used for generating the CDF's

5.4.4 Sampling points per distribution

The following figure shows the effect of changing the number of sample points in each distribution. The standard setup uses 15 sample points taken at equally distanced positions between the 5´th and 95´th percentiles. The data spans between 2 and 20 sampling points where the clustering and classification algorithm is re-run 12 times.

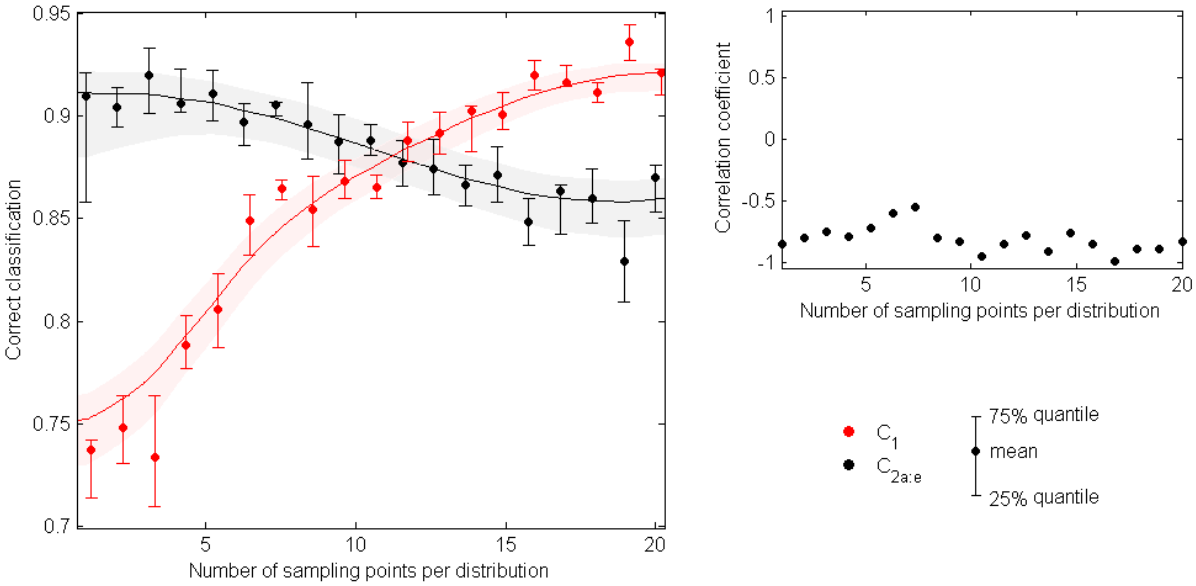


Figure 5.7: Classification result when changing the number of sampling points for each distribution

6 Discussion and Conclusion

Both the elastic map and Kohonen SOM methodologies produce fairly good and almost equivalent 2-dimensional manifold representations of the given data sets. The elastic map methodology gives somewhat better classification results, though, the differences are not extreme and it may be possible that some other system setups will yield better classification results. In addition, though not previously stated, the training phase for the Kohonen SOM is substantially shorter than the corresponding phase for the elastic map (30-60 seconds for Kohonen SOM, 10-30 *minutes* for elastic map).

As for changing the parameters used in the clustering phase it should first be noted yet again that different classification results were obtained each time a clustering was conducted (due to the randomized initiation of the EM-algorithm used to find the signature means) and when rerunning the clustering using the same parameter setup many times, it is possible to calculate Pearson's correlation coefficient. As it turns out, the correlation coefficient is almost always close to -1. What this implies is that if a good classification result were obtained for class C_1, it was bad for C_2 and vice versa.

When changing the exponential-decay parameter to a low value, the signature output from the manifold are, in the extreme, reduced to a single point. When this stage is approached, class C_1 has low classification correctness while it is high for C_2. However, the variance in the obtained classification results is high for both classes. In the other extreme, when increasing the exponential-decay-parameter to a large value, the manifold output is "smoothed" and blurred. The classification correctness for C_1 is here smaller than that of C_2, but the difference is less extreme and the variance in the classification results is, for both classes, fairly small. In the middle section, when the exponential-decay parameter varies between approximately 2 and 10, the classification for class C_1 is higher than that for C_2 and the variance is also smaller for C_1.

Why this effect is seen is unknown. It may be due to class C_1 being much more widely separated in the high-dimensional space as compared to C_2, meaning that the behavior for C_2 is much more "typical" when compared to itself while C_1 is more erratic, possibly consisting of several sub-behaviors or such. It may also be simply due to bad choices of statistical values.

The ´best´ exponential-decay parameter to choose is obviously dependent on the importance of identifying any of the two classes, but if it is assumed that both should be classified approximately equivalently, the best parameter would lie somewhere between 8 and 15.

Changing the number of neighbors considered in the k-NN algorithm does show interesting effects when changing from 1-2 to 5+ neighbors. When considering a single neighbor, the classification is better for C_1 while the effect is more or less indistinguishable when increasing to 5 or more neighbors. However, when increasing the number of neighbors, the classification variance increases substantially. It appears that the best k-NN setup would be to use between 5 and 10 neighbors.

The effect of using different length of the time series, both for feeding the trained system as well as for training the system itself, is quite interesting. What the figure shows is that the classification correctness, for both classes, increases as the length of the time series increases, and it continues to increase regardless of the length of the times series used to warp the manifold. In fact, the figures for the 30-, 45-, 60-, 90- and 120-second are almost equivalent. What it implies is that the length of the time series used to warp the manifold is of no great importance; The same

classification result is obtained whether 45-second or 120-second CDF's are used (30-second CDF's does show some lower classification result for class C_1). This does imply that there truly is convergence in distribution (which is further implied by the greater classification result when using longer time series to feed the trained systems).

The effect of changing the number of sampling points per distribution is shown in figure 5.7. It shows that the classification is ´best´ at around 10 sampling points per distribution. It also shows that better classification results for C_1 is obtained when increasing the number of sampling points, whereas the classification correctness decreases for C_2.

7 Future work

The most obvious additions that can be made for any future work is to simply change the statistical values under consideration. Those used in this project are chosen fairly arbitrarily and only vaguely theoretically justified. However, the theoretical justification is not necessarily the best method to find statistical values that can distinguish between these classes in particular – *any* statistical value that can do so is justified for use, regardless of it having steady theoretical grounds. The future work that can be done is thus to test a wide range of statistical quantities to see whether they yield any desirable results, for example by fitting a manifold to sampled points from the distribution and see if the classes do separate.

Another addition that can be made regards the preprocessing of the data. The problem is that the given data contains lots of empty holes. These missing data points are filled in using simple linear interpolation, but this is likely not the best method. For example, this interpolation method will fill the empty holes equivalently, regardless of any past or future behaviors of the time series except the nearest updates, thus making the interpolation equal for any classes. Using a Kalman filter, for example, would mean that one makes an assumption of an internal state of the observed object, where the internal state is estimated from the observations. Filling the holes would then be equivalent to calculating the object properties based on the best estimate of the internal state at that time, thus making use of much more of the observational data.

References

[1] H. Shaban, S. Tavoularis. “Identification of flow regime in vertical upward air-water pipe flow using differential pressure signals and elastic maps.” In: *International Journal of Multiphase Flow*. (Jan. 2014).

[2] M.O. Afolabi, O. Olude. “Predicting Stock Prices Using a Hybrid Kohonen Self Organizing Map (SOM)”. *Proc. 40th Hawaii International Conf. on System Sciences*. 2007.

[3] J.K. Hunter. *An Introduction to Real Analysis*. University of California at Davis. 2012. pp. 93-94. URL: https://www.math.ucdavis.edu/~hunter/m125a/intro_analysis_ch7.pdf [2015, May 22]

[4] R. Sjamaar. *Manifolds and Differential Forms*. Cornell University. Aug 2006. pp. 67-68. URL: http://www.math.cornell.edu/~sjamaar/papers/manifold.pdf [2015, May 22]

[5] D.B. Chuong, B. Serafim. “What is the expectation maximization algorithm?” In: *Nature Biotechnology 26* (2008), pp. 897 - 899. DOI: 10.1038/nbt1406 URL: http://www.nature.com/nbt/journal/v26/n8/full/nbt1406.html [2015, May 22]

[6] Cambridge University. *K-means*. Cambridge University. 2008. URL: http://nlp.stanford.edu/IR-book/html/htmledition/k-means-1.html [2015, May 22]

[7] O. Sutton. *Introduction to k Nearest Neighbor Classification and Condensed Nearest Neighbour Data Reduction*. University of Leicester. Feb. 2012. URL: http://www.math.le.ac.uk/people/ag153/homepage/KNN/OliverKNN_Talk.pdf [2015, May 22]

[8] C.O.S. Sorzano, J. Vargas, A. Pascual-Montano. *A survey of dimensionality reduction techniques*. Autonomous University of Madrid. Mar. 2014. URL: http://arxiv.org/abs/1403.2877 [2015, May 22]

[9] L.I. Smith. *A tutorial on Principal Component Analysis*. University of Otago. Feb. 2002. URL: http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf [2015, May 22]

[10] A.N. Gorban, A.Y. Zinovyev. *Principal Graphs and Manifolds*. University of Leicester, Institut Curie. May 2011. DOI: 10.4018/978-1-60566-766-9 URL: http://arxiv.org/abs/0809.0490 [2015, May 22]

[11] A.N. Gorban, A.Y. Zinovyev. *Elastic Maps and Nets for Approximating Principal Manifolds and Their Application to Microarray Data Visualization*. University of Leicester, Institut Curie. DOI: 10.1007/978-3-540-73750-6_4 URL: http://arxiv.org/abs/0801.0168 [2015, May 22]

[12] S.M Guthikonda. *Kohonen Self-Organizing Maps*. Wittenberg University. Dec. 2005. URL: http://www.shy.am/wp-content/uploads/2009/01/kohonen-self-organizing-maps-shyam-guthikonda.pdf [2015, May 22]

[13] V. Spruyt. “The Curse of Dimensionality in classification.” In: *Computer vision for dummies.* Apr. 2014. URL: http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/ [2015, May 22]

[14] D. Gamarnik, J. Tsitsiklis. Class lecture. *Lecture 17: Convergence of random variables*. Massachusetts Institute of Technology. Nov. 2008. URL: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-436j-fundamentals-of-probability-fall-2008/lecture-notes/MIT6_436JF08_lec17.pdf [2015, May 22]

[15] J.H. Reif. Class lecture. *Lecture 6: Analysis of Quicksort*. Duke University. Jan. 1999. URL: http://db.cs.duke.edu/courses/spring99/cps130/lectures/lect06.pdf [2015, May 22]