Kernel-based unsupervised trajectory clusters discovery

C. Piciarelli, C. Micheloni, Gian Luca Foresti

Kernel-based unsupervised trajectory clusters discovery

C. Piciarelli, C. Micheloni, G.L. Foresti
University of Udine
Department of Mathematics and Computer Science
{piccia—michelon—foresti}@dimi.uniud.it

Abstract

Nowadays support vector machines (SVM) are among the most popular tools for data clustering. Even though the basic SVM technique works only for 2-classes problems, in the last years many variants of the original approach have been proposed, such as multi-class SVM for multiple class problems and single-class SVM for outlier detection. However, the former is based on a supervised approach, and the number of classes must be known a-priori; the latter performs unsupervised learning, but it can only discriminate between normal and outlier data. In this paper we propose a novel technique for data clustering when the number of classes is unknown. The proposed approach is inspired by single-class SVM theory and exploits some geometrical properties of the feature space of Gaussian kernels. Experimental results are given with special focus on the field of trajectory clustering[1].

1. Introduction

In this paper we propose a kernel-based technique for unsupervised, automatic cluster discovery in data sets where no prior knowledge on the number of clusters is available. The technique has some degree of generality, and thus it can have a number of possible applications, but we primarily developed this work in the context of trajectory analysis for anomaly detection. This assumption mainly influences the type of clusters we are searching for: our aim was to group together trajectories with similar shapes, which is basically a search for groups of patterns with low spatial distances from one another in a proper feature space. This definition of cluster is somewhat stricter than the more general definition of arbitrary-shaped clusters that can be found in other clustering works, e.g. [7].

[1]This work was partially supported by the Italian Ministry of University and Scientific Research under the project "Ambient Intelligence: event analysis, sensor recofiguration and multimodal interfaces".

Trajectory clustering is usually performed by applying standard clustering algorithms to variable-length trajectories on which a similarity measure has been defined, or to trajectories mapped to a fixed-size feature space, depending on the nature of the chosen clustering method. A popular work on trajectory clustering has been done by Stauffer and Grimson [13], where vector-quantization-based codebooks are are clustered with a hierarchical technique. Many other works are based on tools already popular in other time-series analysis problems, such as Hidden Markov Models: Porikli uses eigenvector analysis in the HMM feature space [9]; other works using HMM have been proposed by Alon et al. [2] or by Saunier and Sayed [10]. Hu et al. [5] resample trajectories to a fixed length and group them using a hierarchical spatial/temporal fuzzy k-means clustering; Lin et al. [6] use hierarchical k-means on trajectories' wavelet coefficients; Bennewitz et al. [3] find typical motion patterns using Expectation Maximization; Juneio and Foroosh [1] use normalized-cuts on a graph modelling trajectory similarities based on the Hausdroff distance; Wang et al. [15] proposed a method based on spectral clustering. Note that some of the techniques described above, especially the ones based on partitioning methods such as k-means, rely on prior knowledge on the number of clusters to be searched for.

In the last years, support vector machines have become a very popular clustering tool, although they do not seem to be widely adopted yet for trajectory clustering purposes. Standard SVM are based on a supervised approach, and thus are generally not suitable for practical trajectory analysis applications; single-class SVM can be applied in order to identify outliers as we did in our previous works [8], but they cannot split the group of normal trajectories in smaller clusters. In this paper, we propose a clustering technique based on the kernel methods formalism, especially inspired by theoretical results of single-class SVM. In [8] we used some geometrical properties of the feature space of Gaussian kernels in order to detect outlier trajectories; in this paper the same properties are used in a different way in order to count the number of possible cluster and to assign

trajectories to each cluster.

2. Single-class SVM theory

Support vector machines are a mathematical tool for classification and regression problems that gained a large popularity in the last years due to their robust theoretical foundations and excellent practical results. SVMs extend a mathematically sound optimal linear classification technique, initially proposed by Vapnik et al. [14] in the context of statistical learning theory, to non-linear problems by means of kernels computing dot products in complex feature spaces. In the last years many variants of the basic technique have been proposed, such as multiclass-SVM [4] and single-class SVM [11]. Multiclass Support Vector Machines address the problem of clustering data in three or more groups, but rely on a supervised learning approach, where the number of classes is known a priori. On the other hand, single-class Support Vector Machines are based on unsupervised learning, but they cannot split the "normal data" set in clusters of similar data, since by assumption all the data, with the exception of outliers, belong to a single class. However, in this paper we will show that some geometric properties of the single-class SVM feature space can be used in order to perform data clustering without knowing the real number of clusters.

In order to better understand the remaining sections of this work, a brief description of the single-class SVM theory is here given. For further details on support vector machines, see [12]. Given a set of unlabelled measures generated according to an unknown probability distribution P, single-class support vector machines are used to estimate the support of a particular quantile of P. The goal is to find an appropriate region in the input space $\mathcal{X}$ containing most of the data drawn from P, possibly leaving outliers outside this region. In the SVM framework this can be obtained by implicitly defining a function Φ mapping the data in a new space $\mathcal{H}$ (called *feature space*) and by searching for a linear decision hyperplane in the feature space which maximises its distance from the origin, while only a small fraction of data (the outliers) fall between the hyperplane and the origin. This can be expressed in terms of the constrained minimisation problem

$$\begin{aligned} \min_{\boldsymbol{w},\boldsymbol{\xi},\rho} \ & \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{\nu n}\sum_i \xi_i - \rho \\ \text{subject to } \ & \boldsymbol{w}\cdot\Phi(\boldsymbol{x}_i) \geq \rho - \xi_i, \quad \xi_i \geq 0 \end{aligned} \tag{1}$$

where $\boldsymbol{x}_i \in \mathcal{X}, i \in [1 \dots n]$ are n training data in the input space $\mathcal{X}$, $\Phi : \mathcal{X} \to \mathcal{H}$ is the function mapping vectors $\boldsymbol{x}_i$ in the feature space $\mathcal{H}$ and $(\boldsymbol{w}\cdot\Phi(\boldsymbol{x})) - \rho = 0$ is the decision hyperplane in $\mathcal{H}$ (see also figure 1). In the minimization process, outliers are linearly penalized by the slack variables ξ_i, whose weight is controlled by the parameter $\nu \in (0, 1]$. Introducing the Langrangian multipliers α_i, problem (1) can be reformulated in its dual form

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \ & \frac{1}{2}\sum_{i,j}\alpha_i\alpha_j k(\boldsymbol{x}_i,\boldsymbol{x}_j) \\ \text{subject to } \ & 0 \leq \alpha_i \leq 1/(\nu n) \\ & \sum_i \alpha_i = 1 \end{aligned} \tag{2}$$

where $k(\boldsymbol{x}_i,\boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i)\cdot\Phi(\boldsymbol{x}_j)$ is a kernel function. Values α_i can be found by solving the problem with standard quadratic programming methods; $\boldsymbol{w}$ is given by

$$\boldsymbol{w} = \sum_i \alpha_i \Phi(\boldsymbol{x}_i) \tag{3}$$

and, for any vector $\Phi(\boldsymbol{x}_i)$ with $\alpha_i \neq 0$, the following equations hold

$$\rho - \xi_i = (\boldsymbol{w}\cdot\Phi(\boldsymbol{x}_i)) = \sum_j \alpha_j k(\boldsymbol{x_i},\boldsymbol{x_j}) \tag{4}$$

with $\xi_i > 0$ for outliers and $\xi_i = 0$ for support vectors lying on the decision plane. The decision function in the input space $\mathcal{X}$ is thus defined as

$$\begin{aligned} f(\boldsymbol{x}) &= \operatorname{sign}((\boldsymbol{w}\cdot\Phi(\boldsymbol{x})) - \rho) \\ &= \operatorname{sign}\Big(\sum_i \alpha_i k(\boldsymbol{x},\boldsymbol{x_i}) - \rho\Big) \end{aligned} \tag{5}$$

The solution is sparse since $\alpha_i = 0$ for any $\boldsymbol{x}_i$ lying within the support region identified by the decision function: the majority of the training vectors do not contribute to the definition of the decision function. For the other vectors (called *support vectors*), $0 < \alpha_i < 1/(\nu n)$ if the vector lies on the decision hyperplane, and $\alpha = 1/(\nu n)$ if the vector is an outlier.

The parameter ν has an intuitive meaning since it tunes the number of acceptable outliers. From (1) it can be seen that, when $v \to 0$, the outliers' penalization factors grow to infinity, thus leading to a hard margin solution, where no outliers are allowed at all. On the other hand, if $\nu = 1$, the constraints in (2) allow a single solution in which all the α_i are saturated to the maximum value $1/n$, thus all the vectors are outliers. More specifically, it can be proven that ν is an upper bound for the fraction of outliers and a lower bound for the fraction of support vectors (e.g. with $\nu = 0.1$ at least 10% of the training vectors will be support vectors, and at most 10% will be outliers).

3. Cluster discovery

As described in the previous section, single-class support vector machines work by mapping the data from the input

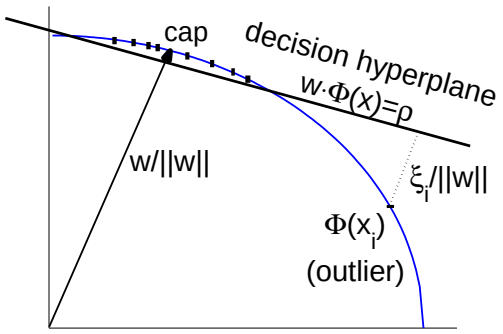


Figure 1. A visualisation of $\mathcal{H}$ in a simple 2D case.

space $\mathcal{X}$ into a feature space $\mathcal{H}$ and by searching for a hyperplane in $\mathcal{H}$ separating the normal data from outliers; in particular, outliers will lie in the negative semispace containing the origin. Even though the feature space is generally very complex and possibly unknown, some hints on its structure can be derived by means of simple geometric considerations. In this section we will highlight some of the most important properties of the feature space that will lead us to the definition of an unsupervised clustering algorithm. In the rest of this paper we will assume to work with Gaussian kernels.

We will first focus on the effect that normalised kernels have on the data distribution in the feature space $\mathcal{H}$. A kernel is considered normalised if $k(x,x) = 1,\ x \in \mathcal{X}$; note that the Gaussian kernel is always normalised, and for any kernel k a normalised version k' exists, defined as $k'(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_1, \boldsymbol{x}_2)/\sqrt{k(\boldsymbol{x}_1, \boldsymbol{x}_1), k(\boldsymbol{x}_2, \boldsymbol{x}_2)}$. By definition of kernel, this implies that, if $k(\boldsymbol{x}, \boldsymbol{y}) = \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{y})$ is a normalised kernel, then the following equations holds:

$$\begin{aligned} \|\Phi(\boldsymbol{x})\|^2 &= \Phi(\boldsymbol{x}) \cdot \Phi(\boldsymbol{x}) \\ &= k(\boldsymbol{x}, \boldsymbol{x}) \\ &= 1 \end{aligned}$$

In other words, when using normalised kernels, the data are mapped on the surface of a hypersphere with unitary radius and centred in the origin of the feature space. The classification hyperplane computed by single-class SVMs thus cuts the hypersphere such that all the normal data lie on a hyperspherical cap, while outliers lie on the remaining surface of the hypersphere. This observation leads to the intuitive idea that normal data in $\mathcal{H}$ are close to one another, while outliers lie far from the cluster of normal data (figure 1).

This notion will be crucial for the definition of our approach, but there is no general way to measure distances in $\mathcal{H}$, since its structure is generally unknown. However, in the special case of normalised kernels, all the data lie on the surface of a hypersphere, and thus a good distance measure between two elements $\Phi(\boldsymbol{x}_1)$ and $\Phi(\boldsymbol{x}_2)$ is given by the angle $\theta = \widehat{\Phi(\boldsymbol{x}_1)O\Phi}(\boldsymbol{x}_2)$. By using again the properties of normalised kernels, we have

$$\begin{aligned} \Phi(\boldsymbol{x}_1) \cdot \Phi(\boldsymbol{x}_2) &= \|\Phi(\boldsymbol{x}_1)\| \|\Phi(\boldsymbol{x}_2)\| \cos(\theta) \\ &= \cos(\theta) \end{aligned}$$

and thus θ can be computed as

$$\begin{aligned} \theta &= \arccos\left(\Phi(\boldsymbol{x}_1) \cdot \Phi(\boldsymbol{x}_2)\right) \\ &= \arccos\left(k(\boldsymbol{x}_1, \boldsymbol{x}_2)\right) \end{aligned} \tag{6}$$

Let now define the mean $\boldsymbol{m}$ of all the n data in $\mathcal{H}$ as

$$\boldsymbol{m} = \frac{1}{n} \sum_{i=1}^{n} \Phi(\boldsymbol{x}_i)$$

and its projection on the unitary hypersphere $\boldsymbol{m}/\|\boldsymbol{m}\|$. The angle θ_x between any data $\Phi(\boldsymbol{x})$ and $\boldsymbol{m}/\|\boldsymbol{m}\|$, according to eq. 6, is defined as

$$\begin{aligned} \theta_x &= \arccos\left(\frac{\boldsymbol{m} \cdot \Phi(\boldsymbol{x})}{\|\boldsymbol{m}\|}\right) \\ &= \frac{\sum_i k(\boldsymbol{x}, \boldsymbol{x}_i)}{\sqrt{\sum_i \sum_j k(\boldsymbol{x}_i, \boldsymbol{x}_j)}} \end{aligned} \tag{7}$$

We could have reached the same result in terms of single-class SVM theory by defining θ_x as the angle between outliers and the centre $\boldsymbol{w}/\|\boldsymbol{w}\|$ of the hyperspherical cap and by noting that, according to eq. 3, $\boldsymbol{w} \to \boldsymbol{m}$ when $\nu \to 1$ (full details are given in [8]). In [8] we used eq. 7 to automatically detect the number of outliers; in this paper instead we will focus on cluster discovery and thus we will assume that no outliers are present; the integration of the two techniques for cluster discovery in presence of outliers is straightforward.

In order to better understand the rest of this section, it is helpful to give an intuitive interpretation of eq. 7. First of all note that the denominator is a constant term. Then, observe that the numerator term actually is a Parzen window density estimator with kernel k. If the kernel is a Gaussian one, this implies that the more elements are close to $\boldsymbol{x}$, the higher the numerator will be, and consequently θ will assume low values. In other words, zones with high densities in the input space $\mathcal{X}$ will all be mapped near $\boldsymbol{m}/\|\boldsymbol{m}\|$ in $\mathcal{H}$ (this explains how single-class SVM can fit in the same class groups of data apparently distant and separated in the input space); at the same time, elements in zones with low densities will have a larger θ and will be mapped near the boundaries of the hypersphere region containing the mapped data (this region does not coincide with the full hypersphere: if the kernel is always positive as in the Gaussian case, then $0 \leq \theta = \arccos(k(x_1, x_2)) \leq \pi/2$ for any $\boldsymbol{x}_1, \boldsymbol{x}_2$, and thus

all the data are forced to lie on a portion of the hypersphere subtended by an angle of $90°$ maximum).

Let now A, B be two distinct clusters in $\mathcal{X}$, where a cluster is defined as a group of elements with high density as defined by the Parzen window estimator. Then, the elements belonging to the two clusters will all be close to $\boldsymbol{m}/\|\boldsymbol{m}\|$ (will have a small θ). Our algorithm computes the angle θ_x for each $\boldsymbol{x} \in A, B$ and searches for the element $\boldsymbol{x}_{max}$ with highest θ; this element is then removed from the training set. Let assume that $\boldsymbol{x}_{max} \in B$: in this case its removal will cause a small decrease of the density of all region containing B, and thus the entire set B will slightly move far from $\boldsymbol{m}/\|\boldsymbol{m}\|^2$. Because of this, if we repeat the procedure there is a good chance that the next $\boldsymbol{x}_{max}$ belongs again to B (although this is not guaranteed). By iterating the procedure, B will thus shift farther and farther from $\boldsymbol{m}/\|\boldsymbol{m}\|$, and $\theta_{x_{max}}$ will increase at each iteration. Only when the last element of B is removed, $\theta_{x_{max}}$ will drop to smaller values, since only A will be present in the data set. At that point, removing elements from A will cause a progressive decrease of the values of $\theta_{x_{max}}$ until $\theta_{x_{max}} = 0$ when a single element is left in the data set. Note that the above procedure is not limited to the case of two clusters, but can be generally applied with any number of clusters. The pseudo-code for the whole algorithm is given in table 1, where the procedure is expressed in terms of operations on the Gram matrix K (this is, $K_{i,j} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$). Its time complexity is $O(n^3)$, since for each one of the n data elements a fixed number of operations are performed on the Gram matrix, whose size is $n \times n$; the algorithm is thus not well suited for massive data sets. Observe however that the code has a very simple implementation: the algorithm has been inspired by SVM theory, but no SVM must actually be trained, thus avoiding the complex solution of quadratic optimisation problems.

$\boldsymbol{K}$ = compute $n \times n$ Gram matrix
for s=1:n
$denom = \sqrt{\sum_i \sum_j \boldsymbol{K}_{i,j}}$
$num = \min_i \left(\sum_j \boldsymbol{K}_{i,j} \right)$
$\theta_{x_{max}}(s) = \arccos\left(\frac{num}{denom}\right)$
remove column i and row i from $\boldsymbol{K}$
endfor

Table 1. Pseudo-code for the computation of $\theta_{x_{max}}$ values.

The procedure here described gives a clear hint for the definition of a cluster detection procedure: by observing

[2] As a side note, this implies that the function Φ depends on the data set: removing an element from the set will also change the mapping function.

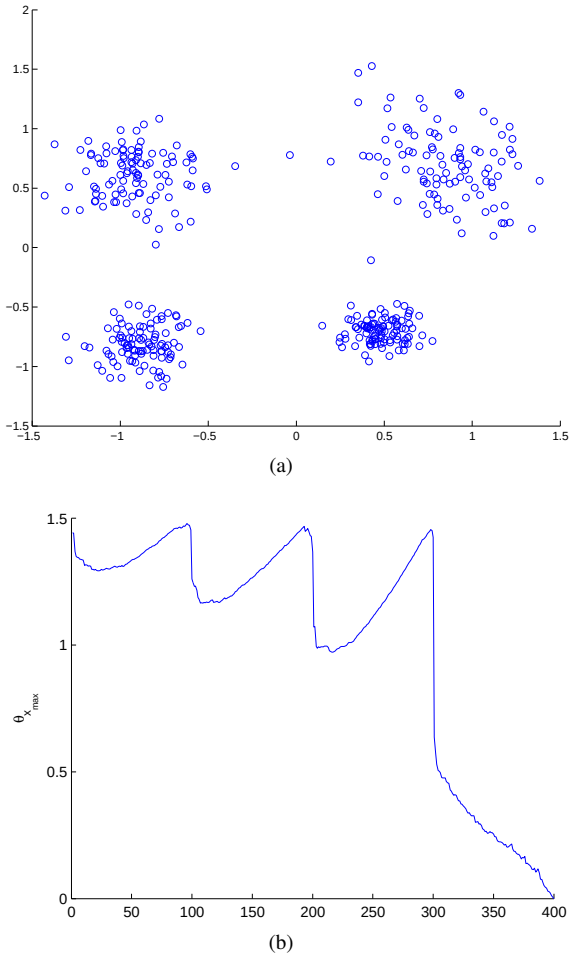


Figure 2. Detecting clusters by analysing $\theta_{x_{max}}$. (a) the training set; (b) values of $\theta_{x_{max}}$ as data are removed from the training set

.

subsequent values of $\theta_{x_{max}}$, a typical pattern should be detected, with $\theta_{x_{max}}$ values progressively increasing as the data are removed from a cluster, and with sudden drops when the last element is removed from a cluster. Experimental results confirm this prediction (see figure 2).

Detecting clusters is now a simple matter of searching for points where $\theta_{x_{max}}$ decreases significantly (we call them discontinuity points). The number of clusters is the number of discontinuity points plus one, while all the data whose $\theta_{x_{max}}$ lies between two discontinuity points belong to the same cluster. As already pointed out, there is no theoretical guarantee on the correctness of the final result, since the proposed method is based on a heuristic which could possibly fail; in the next section experimental results will be given in order to measure the algorithm performances and prove the strength of the heuristic.

4. Experimental results

Since the proposed algorithm has been developed in the context of a trajectory analysis application for anomaly detection, experimental results will be based on trajectory clustering. In order to perform testing on a large data set, a trajectory generator has been written in Matlab. Given a specified number of clusters and the number of trajectories in each cluster, the trajectory generator creates a set of noise-perturbed 2D trajectories grouped in the required number of clusters; each trajectory is composed by 16 coordinate pairs normalised in the $[-1,1]\times[-1,1]$ interval. The generator avoids the creation of totally overlapping clusters, since they would influence negatively the error measurement process.

We considered the problem of cluster discovery in several cases, ranging from 1 to 10 real clusters in the data set, where each cluster is composed by 50 trajectories. For each case, 50 experiments with random data sets have been performed, in order to have statistically meaningful results; figure 4 shows some examples of the generated data sets and their classification, together with the corresponding plot of $\theta_{x_{max}}$. All the tests have been performed using a Gaussian kernel with $\sigma = 1$. The test results in terms of number of detected clusters are summarised by the confusion matrix in table 2: each row represents the real number of clusters in the data sets, while columns show how many clusters were actually detected in the 50 runs. As it can be seen, the performances seem to degrade linearly with the number of clusters; as an overall result, more than 85% of the tests resulted in correct classification, and more than 98% resulted within an error of ± 1 cluster.

real	detected										
	1	2	3	4	5	6	7	8	9	10	11
1	50	0	0	0	0	0	0	0	0	0	0
2	0	49	1	0	0	0	0	0	0	0	0
3	0	1	47	2	0	0	0	0	0	0	0
4	0	0	3	45	2	0	0	0	0	0	0
5	0	0	0	3	46	1	0	0	0	0	0
6	0	0	0	0	1	43	4	2	0	0	0
7	0	0	0	0	0	4	41	5	0	0	0
8	0	0	0	0	0	2	3	38	6	1	0
9	0	0	0	0	0	0	1	6	35	7	1
10	0	0	0	0	0	0	0	4	7	32	7

Table 2. Confusion matrix: existing clusters (rows) compared to detected clusters (columns).

Performance measurement in terms of number of clusters is essential if the proposed technique is used as a "cluster counter" for those clustering algorithms requiring the number of clusters to be known a priori, but it is not sufficient if the algorithm is used as a clustering tool itself. In this case, the number of trajectories not assigned to the correct cluster should be measured as well. We considered the data sets where the correct number of clusters has been detected in the previous test, and we measured the number of misclassified trajectories. Results are given in table 3 as a percentage of erroneously classified trajectories over the total amount of trajectories in a given test set. For example we can see that, whenever the system correctly detects the presence of 10 clusters, approximately 7.1% of the trajectories are assigned to the wrong cluster.

clusters	errors
1	0%
2	0.4%
3	0.96%
4	1.62%
5	2.24%
6	3.52%
7	4.15%
8	5.12%
9	6.92%
10	7.10%

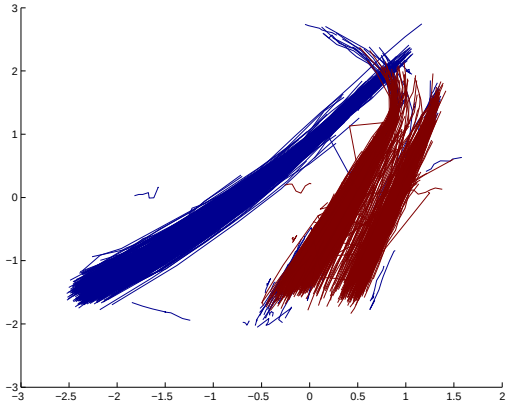
Table 3. Amount of misclassified trajectories when the correct number of clusters is detected.

Finally, we also tested the system on real-world data, although in this case performances cannot be easily measured, because of the lack of objective ground truth validation data. Since real trajectories have different lengths and are perturbed by noise, a pre-processing step is applied, in which trajectories are smoothed and subsampled to a fixed number of coordinate points (16 as in the previous tests). Figure 3 shows an example of vehicle trajectories extracted from a 51-minutes-long video sequence of an urban road, during which 1430 vehicles were detected by means of multi-gaussian background-subtraction techniques and tracked using a combination of Kalman-based and CamShift trackers. In this case, the system has correctly detected the two main traffic flows.

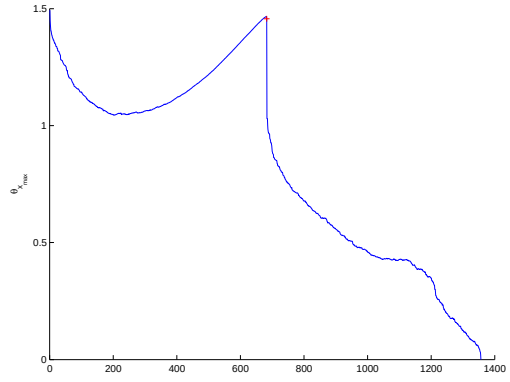
5. Conclusions

In this paper we presented a novel approach to cluster discovery. Many popular clustering algorithms, such as K-Means of Expectation Maximization, require the number of clusters to be known a priori, which is often a strong limit for practical use. By taking inspiration from support vector machines theory and by exploiting the geometric properties of the kernel feature space, we proposed a technique that can automatically identify the number of clusters, as well as

(a)



(b)



(c)

Figure 3. Application of the proposed algorithm to real-world trajectories. (a) the monitored scene of an urban road; (b) the two main clusters detected by the system; (c) the corresponding θ values.

the clusters themselves. Even if not well-suited for massive data sets because of its $O(n^3)$ time complexity, the algorithm has given good results on datasets with small amounts of clusters. As a future step, we plan to integrate the algorithm with our anomaly detection technique for trajectory analysis.

References

[1] Euclidean path modeling for video surveillance. *Image and vision computing*, 26:512–528, 2007.

[2] J. Alon, S. Sclaroff, G. Kollios, and V. Pavlovic. Discovering cluster in motion time-series data. In *Computer Vision and Pattern Recognition*, pages 375–381, Madison, WI, USA, 2003.

[3] M. Bennewitz, W. Burgard, and S. Thrun. Learning motion patterns of persons for mobile service robots. In *Proc. of IEEE International conference on robotics and automation*, volume 4, pages 3601–3606, 2002.

[4] E. Bredensteiner and K. Bennett. Multicategory classification by support vector machines. *Computational Optimization and Applications*, 12:35–46, 1999.

[5] W. Hu, X. Xiao, Z. Fu, D. Xie, T. Tan, and S. Maybank. A system for learning statistical motion patterns. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28(9):1450–1464, 2006.

[6] J. Lin, M. Vlachos, E. Keogh, and D. Gunopulos. Iterative incremental clustering of time series. In *Int. Conf. on Extending Database Technology*, pages 106–122, Crete, Greece, 2004.

[7] G. Nosovskiy, D. Liu, and O. Sourina. Automatic clustering and boundary detection algorithm based on adaptive influence function. *Pattern Recognition*, 41:2757–2776, 2008.

[8] C. Piciarelli and G. Foresti. Anomalous trajectory detection using support vector machines. In *IEEE Int. conf. on Advanced Video and Signal-based Surveillance*, London, UK, 2007.

[9] F. Porikli. Trajectory pattern detection by hmm parameter space features and eigenvector clustering. In *8th European Conference on Computer Vision*, Prague, CZ, 2004.

[10] N. Saunier and T. Sayed. Clustering Vehicle Trajectories with Hidden Markov Models. Application to Automated Traffic Safety Analysis. In *International Joint Conference on Neural Networks*, Vancouver, 2006.

[11] B. Schölkopf, , J. Platt, J. Shave-Taylor, A. Smola, and R. Williamson. Estimating the support of a high-dimensional distribution. Technical Report TR 87, Microsoft Research, Redmon, WA, 1999.

[12] B. Schölkopf and A. Smola. *Learning with Kernels — support vector machines, regularization, optimization, and beyond*. MIT press, 2002.

[13] C. Stauffer and W. Grimson. Learning patterns of activity using real-time tracking. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22(8):852–872, 2000.

[14] V. Vapnik and A. Lerner. Pattern recognition using generalized portrait method. *Automation and Remote Control*, 24:774–780, 1963.

[15] X. Wang, K. Tieu, and E. Grimson. Learning semantic scene models by trajectory analysis. In *European Conference on Computer Vision*, pages 110–123, Graz, A, 2006.

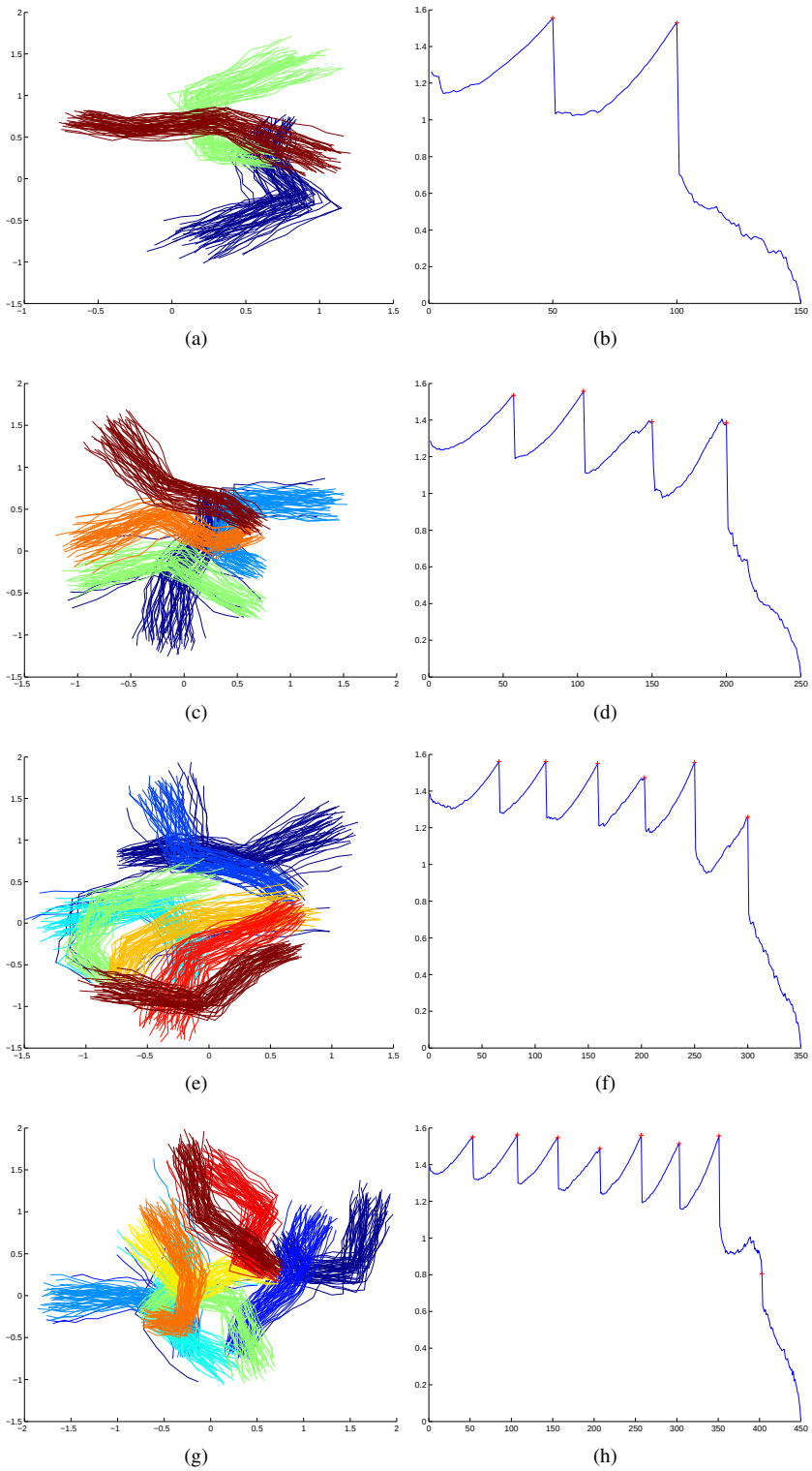


Figure 4. (a),(c),(e),(g): some sample data sets with 3, 5, 7 and 9 clusters, trajectory colours are assigned on the basis of the computed clustering; (b),(d),(f),(h): the corresponding θ plots, where the discontinuity points have been marked by a red $+$ sign.