Masters Thesis Lukas Kohlhase

Lukas Kohlhase

October 21, 2017

Abstract

[1] I am an abstract that currently does not contain any content. EdN:1

[2] EdN:2

[1]EdNote: Make Abstract

[2]EdNote: change Title to something reasonable, still have to do good titlepage as well

1 Introduction

Okay we want some motivation stuff here.

Why do we want to segment data? (Useful for learning stuff, making datasets etc.) Why do we wnat to do so automatically etc. [3] EdN:3

1.1 Related Work

Standard related work stuff here [4] EdN:4

[5] SFA is nice, for reasons (universality), and it is also an unsupervised technique to get some classification data from a timeseries. Thus we want to try to use it for segmentation of timeseries data as well. EdN:5

[3]EdNote: Make Introduction

[4]EdNote: Start Related Work here

[5]EdNote: Make Transition nicer

2 Basic Method

[6] EdN:6

Paragraph about why SFA is good here.

Paragraph about why we want to use selfsimilarity. (Motions are made of many oft repeated motions)

Thus our goal is to investigate an algorithm that segments motion data using SFA and using the principle of self similarity to decide where to place good boundary points segmenting the data.

To do this we first apply SFA to the data, then construct a fitting similarity matrix from the gained slow features, apply spectral clustering to this similarity matrix, and finally find boundary points based on this clustering. [7] [8] EdN:7 EdN:8

2.1 Slow Feature Analysis

[9] We base our description on SFA on [10]. EdN:9

SFA is based on the principle that meaningful, global change in data happens slowly, compared to the possibly quick oscillations that local sensors can identify. If we were to look at a video of a zebra running in front of the camera, local sensors would quickly detect large changes in the color, due to the zebras stripes, however the global movement of the zebra would be much slower. EdN:10

[11] EdN:11

Suppose we have a k-dimensional input signal $x(t)$.

The first step of SFA is to normalize the data $x(t)$ to $\tilde{x}(t)$, where $\tilde{x}(t)$ has 0 mean and unit variance.

Then we expand $\tilde{x}(t)$ with some nonlinear functions $h(x)$. A standard choice for this are all monomials of degree 1 and 2. This is done so that SFA can also recognize nonlinear relationships in the data.

As the last step of preprocessing, we whiten the expanded $z(t) = h(\tilde{x}(t))$ to receive $\tilde{z}(t)$, with again 0 mean and with unit covariance matrix $\tilde{z}\tilde{z}^T = I$. [12] EdN:12

As an intermediary step, we make the matrix $Z = \dot{\tilde{z}}\dot{\tilde{z}}^T$, which is just the covariance matrix of the temporal difference vectors $\dot{\tilde{z}}(t) = \tilde{z}(t+1) - \tilde{z}(t)$.

Finally we apply PCA [13] to Z to get a list of eigenvectors and eigenvalues v_i and λ_i respectively. We use these to define our slow features g_i, with $g_i = v_i^T \tilde{z}(t)$ and $\lambda_1 \geq \lambda_2 \leq \ldots \geq \lambda_k$. EdN:13

We note that we use the smallest eigenvalues instead of the largest eigenvalues, since we want the slowest changing features.

[6] EdNote: We need to define at some point what exactly we're looking for in terms of boundary points. Define input data, and then that we need cutting points

[7] EdNote: Is this enough of a transition?

[8] EdNote: Somewhere we need to get the dimensionality reduction as preprocessing in.

[9] EdNote: Find Paper to cite for slow feature analysis: Probably using this one: http://ieeexplore.ieee.org/document/6790128/

[10] EdNote: add citation here

[11] EdNote: Add part about conditions for problem that SFA solves

[12] EdNote: Possibly describe how whitening is done, it's just PCA again

[13] EdNote: check whether this is first occurrence of PCA, do we need to define what PCA is?

We can then use these feature values to construct a similarity matrix, that we can later use for clustering.

2.2 Make Similarity Matrix

[14] The first step in making a similarity matrix is turning our sequence of feature values $g_1(t), g_2(t), \ldots, g_k(t)$ into a sequence of feature vectors. These feature vectors will then be compared to each other and their similarity scores will make up our similarity matrix. EdN:14

Our first choice is thus how are feature vectors are composed [15], most notably how many features we use and how many timesteps we are comparing at once. EdN:15

The choice of how many features to consider obviously depends on the data that we use, if we expect there to be many relevant features, maybe because there are many semantically relevant slow changing parts of the sequence, e.g. there are many people running in a video, we should be using a high number of features in our feature vectors. If the data is relatively simple or we have a limited amount of computing power, then a lower number might be more appropriate. [16] For our experiments, we defaulted to using 5 features. [17] EdN:16

After choosing the number of features to consider, we must consider how many time steps to incorporate in our feature vectors. This will again be affected [18] by our data, if we expect the segments to be about 100 points long, then using 50 time steps will cover a much larger portion of a segment than if we expected segments to be 1000 points long. [19] Four our experiments, we defaulted to using time steps of [20] 20, which seemed to work well regardless of data structure. EdN:17 EdN:18 EdN:19 EdN:20

Once we have chosen the number $n_{feature}$ of features and the number n_{time} of time steps to consider, we construct our feature vectors $v(t)$ using:

$$v(t) = \begin{pmatrix} g_1(t) \\ g_2(t) \\ \vdots \\ g_{n_{feature}}(t) \\ g_1(t+1) \\ \vdots \\ g_{n_{feature}}(t+n_{time}) \end{pmatrix} = \begin{pmatrix} G_{n_{feature}}(t) \\ G_{n_{feature}}(t+1) \\ \vdots \\ G_{n_{feature}}(t+n_{time}) \end{pmatrix}$$

[21] The simplest way to make a similarity matrix is to simply take the scalar product as a similarity measure. However this has the issue that the resulting EdN:21

[14]EdNote: Think about possibly explaining closer what a similarity matrix is. But it's not really a well defined formal concept

[15]EdNote: This sounds dumb

[16]EdNote: This entire paragraph sounds dumb. Look at it later

[17]EdNote: Is using we for authors proper in non math papers?

[18]EdNote: affect or effect? I think it's affect

[19]EdNote: Is it good to be specific here?

[20]EdNote: time steps or timesteps?

[21]EdNote: lko:Is the G stuff understandable? Or do I specifically need to define it?

scores are not normalized, and that it is not independent of the scaling of vectors. If we take two vectors v and λv with $\lambda \leq 1$, then $v * v \geq \lambda v * \lambda v = \lambda^2 v * v$, even though both are identical.

To remedy this, we use $e^{-d(v,v^*)/\Delta}$ as our similarity measure, for some vectors $v, v*$, some distance measure d, and some scalefactor $\Delta > 0$. This has the property that we only get similarities from $0-1$, and that we get a score of 1 iff [22] the two vectors are identical. The issue with different scaling described for the previous points might still apply, however we are more interested in similar points, so this should be a minor issue [23]. EdN:22 EdN:23

The choice of Δ depends on the given data and previous parameters chosen. If we choose a very small Δ, then most of the similarities will be forced to go to 0, but if we make it too large we will get many results that are too close to 1. A good choice of Δ ensures that the the interval $[0,1]$ is filled relatively evenly. [24] EdN:24

Thus we are left with a similarity matrix S with $(S)_{i,j} = e^{-d(v(i),v(j)/\Delta}$, where the last relevant choice is the choice of distance measure d.

2.2.1 DTW versus Euclidean

The main distances that we used were [25] using the simple euclidean distance and using the Digital Time Warping (DTW) distance. EdN:25

From a purely theoretical perspective, DTW would seem more appropriate, since it takes into account possible misalignment of two sequences. However it has the heavy drawback that computing d_{dtw} the DTW distance between two sequences is $O((n_{features} * n_{time})^2)$, while euclidean distance can be computed in $O(n_{features} * n_{time})$ [26]. In practice it was not uncommon for the larger matrices to take about a minute to compute with using Euclidean distance and taking over 10 hours to compute using DTW.[27] EdN:26 EdN:27

Thus when considering using DTW, we use several tricks to reduce the time needed for computation.

We first use some insight about the structure of our feature vectors $v(t)$. Notably we use that our vectors $v(t)$ consist of subvectors $G(t) = (g_1(t) \dots g_{num_{feature}}(t))$. Since g_i are separate orthogonal features, we expect that the transition from $g_{num_{feature}}(t)$ to $g_1(t+1)$ would have no similarities. [28] Thus it is theoretically sound to use $d_{dtw}(v_(t), v^*(t0) = \sum_{i=0}^{num_{feature}} d_{dtw}(G_i(t), G_i^*(t))$, i.e. we are only aligning the same features to each other. This reduces the complexity to $O(num_{feature} num_{time}^2)$, which is a significant gain. EdN:28

[22]EdNote: lko:Do we write this out? or leave it as iff
[23]EdNote: Is this understandable
[24]EdNote: Put in picture of matrix done with basic euclidean matrix
[25]EdNote: look for dtw paper
[26]EdNote: check whether its big o or small o
[27]EdNote: dtw stuff here. Do I have to define dtw?
[28]EdNote: Ehh this doesn't really work. I want something like any DTW path would go diagonally here

The final trick that we use is that we are mostly interested in local similarities, as we have to find boundary points that separate neighboring clusters and thus it is not as important how similar two feature vectors at the end and the start are, it is much more important to know how similar close points are. Thus the idea is that for close vectors, we use the superior DTW distance, and for far away vectors, we use euclidean distance, or even set them to 0 altogether.

[29] A useful property of DTW distance that we use here is that $d_{dtw}(x, y) \leq d_{euclidean}(x, y) \forall x, y$, since the DTW path would at worst be just the diagonal [30]. Thus $e^{-d_{dtw}(x,y)} \geq e^{d_{euclidean}(x,y)}$, i.e. we possibly make distant points less similar than they should be. This is potentially an upside, as it reduces the chance of clustering techniques giving distant points the same clustering, which can make finding accurate boundary points difficult. EdN:29 EdN:30

However the first step to being able to find a boundary is clustering in the first place. We do this using Spectral Clustering

[31] [32] EdN:31 EdN:32

2.3 Spectral Clustering

[33] We choose to use spectral clustering because it is simple to implement, is based on standard linear algebra, similar to SFA, and because initial testing with other clustering techniques such as k-means clustering did not lead to good results. [34] We follow [35] in their description of spectral clustering. EdN:33 EdN:34

There are several different variants of this technique, varying in the choice of similarity matrix and whether they normalize eigenvectors/values or not. However at their heart, they all follow the same sequence of steps. EdN:35

Given a similarity matrix $S \in \mathbb{R}^{TT}$ and a number k of desired clusters, we first start by computing the unnormalized Laplacian $L = D - S$, where D is the diagonal of S.

Next we compute the k eigenvectors corresponding to the smallest k eigenvalues [36] $u_1, \ldots, u_k$ of L, which we use to construct the matrix $U \in \mathbb{R}^{T \times k}$, which has the eigenvectors u_i as columns. EdN:36

We define y_i to be the vector corresponding to the i-th row of U. This corresponds to taking the i-th elements of the first k eigenvectors [37]. EdN:37

[29] EdNote: Iko: I have to rethink whether I want to use standard euclidean distance, or piecewise taxicab. I know what's meant

[30] EdNote: Iko: not happy about this but have raid now.

[31] EdNote: Iko: Put picture of matrix here. Still need to make it though. Near the diagonal dtw, far away from the diagonal euclidean

[32] EdNote: We probably want pictures of matrices here. Not sure if based on toydata or based on actual data

[33] EdNote: Motivation for spectral clustering here. Not sure about it tbh, it's just the standard approach to use.

[34] EdNote: Iko: pretty ehhhh on this sentence tbh

[35] EdNote: citation here I guess

[36] EdNote: Iko: Do we repeat k here? Not sure if necessary or stylitically nice

[37] EdNote: Iko: In principle this is not well defined, but we explained which vectors we were taking earlier, so it should be fine

Then we use k-means clustering to cluster these [38] y_i, into clusters C_j. EdN:38

We are left with clusters $C_1, \dots, C_k$ that assign each feature vector $v(t)$ an integer representative of their cluster. However, this clustering could be relatively arbitrary, it is common that we get assign e.g. time 1 to cluster 1, time 2 to cluster 2, but time 3 to cluster 1 again. We are looking to segment our time series data, i.e. we need to find points that act as boundaries for a clustering, while minimizing errors.

2.4 Find decision boundaries

As always when minimizing an error, we first have to define an error function.

To do this we first have to decide what error we want to minimize. There are two potential ways to approach this, we can try to minimize error in the similarity matrix or minimize error in the clustering. [39] EdN:39

If we were to minimize errors in the similarity matrix, we would need to find boundary points that result in the smallest and least amount of misclassifications, if we consider every point as represented by their vector in the similarity matrix, or even as represented by just the values of their slow features.

The other option would just require that we represent every point by the cluster-number assigned by spectral clustering, and try to minimize the number of misclassifications [40] that would result in a specific boundary. EdN:40

We go with the second option, since even finding a good error function for the first would be tricky, and would come too close to making the effort of spectral clustering useless [41] EdN:41

Thus we define our error function for the second interpretation. We consider boundary points $b_1, \dots, b_k$ with the accompanying function $g(t, b_1, \dots, b_k)$ assigning the point in time t the value j if j is the most common clustering in the interval $[b_t, b_{t+1})$ and the function $c(t) = j$ if $y_t \in C_j$. For convenience sake we assume that the center of C_i is lower than the center of C_j iff [42] $i < j$. Then we get the error function [43] EdN:42 EdN:43

$$E(b_1, \dots, b_k) = \sum_{i=1}^{T} \delta_{g(t),c(t)}$$

where $\delta_{i,j}$ is the Kronecker delta [44]. Thus for a set of boundary points, we are simply checking how many points would be misclassified if we compare the EdN:44

[38] EdNote: Iko: Do we need to explain k-means clustering? I might want to do it later. Not sure at all. It's a very basic clustering technique.

[39] EdNote: Iko: This whole sequence is bad imo

[40] EdNote: Iko: is misclassifications a word?

[41] EdNote: Okay this entire sequence is fucky, and needs to be reworked or removed. Might be a discussion worth having, so I'm writing something on it, but I'm just leaving it in because removing is easier than adding

[42] EdNote: Iko: again check iff

[43] EdNote: check if T is the appropriate delimitor

[44] EdNote: check if spelling is correct

clustering we get from spectral clustering to the clustering imposed by these boundary points.

This leads us to the optimization problem of finding the minimum $b_1^*, \ldots, b_k^*$ with

$$b_1^*, \ldots, b_k^* = \underset{b_1, \ldots, b_k}{\arg\min}\, E(b_1, \ldots, b_k)$$

We can immediately see that if we only need one or two boundary points, it's viable to just try out all options, however the cost increases exponentially with the amount of boundary points.

How many boundary points are needed is one of the key problems that we work around in the various specific implementations of the entire algorithm that we use. [45] [46] [47] [48]

EdN:45
EdN:46
EdN:47
EdN:48

[45]EdNote: Iko: Dis transition is bad and I should feel bad. But transitions are what I'll need to fix when I have everything written, so YOLO

[46]EdNote: Maybe put a summary in here somewhere?

[47]EdNote: Potentially put reducenumber stuf in here

[48]EdNote: Transition

3 Specific Details

In the previous section, we discussed the core steps of our algorithm, but we left out specific implementations. [49] EdN:49

[50] In our testing, three different variants of the basic algorithm crystallized, which essentially differed in how online the approach was. EdN:50

We will first describe the fully offline approach, then the intermediate approach, and finally the fully online approach.

3.1 Batch

The basic idea behind the fully offline approach is very simple, given some sequence of input data $x(1), \dots, x(T)$ with $x(t) \in \mathbb{R}^n$, we first generate k slow features $g_i(t)$ using SFA on the entirety of our data, then we make feature vectors $v(t)$ as described previously [51], use spectral clustering to cluster the data into $k_{cluster}$ different clusters, and then find the decision boundaries. EdN:51

This short description leaves two major questions open: How do we find decision boundaries and How many different clusters are we looking for?

We will first look into how we could answer the second question. To answer it, we have to define what we regard as a good result of finding boundaries, especially if we use a variable number of boundaries. Intuitively, the most important thing is finding the correct boundaries potentially finding some extraneous [52] is not that big of a deal [53]. Thus when looking at test data to answer the second question, we would look at the best decision boundaries, and see how close they are to the real decision boundaries. EdN:52 EdN:53

Thus, to have a chance at answering the second question, we must be able to find decision boundaries in the first place. Hence we have to first answer the first question. [54] EdN:54

Given a list $c_1, \dots, c_T$ of length T [55] of integers from 1 to $k_{cluster}$, and a list of true boundary points $b^*_1, \dots, b^*_{k_{true}}$ Our goal is to use only $c_1, \dots, c_T$, to find $k_{cluster}$ boundary points $b_1, \dots, b_{k_{cluster}}$ with $b_1 < b_2 < \dots < b_{k_{cluster}}$ so that b_i are close to b^*_j. We consider the best b_i for each b^*j, even if $b_{true} > b_{cluster}$. [56] EdN:55 EdN:56

Clearly without access to b^*, our best bet is to minimize some error function using only c_i. We had previously defined an error function

$$E(b_1, \dots, b_k) = \sum_{i=1}^{T} \delta_{g(t), c_t}$$

[49]EdNote: Iko:I really do not have this transition stuff down atm

[50]EdNote: The flow here should be: First batch, then minibatch as answer to the question: Into how many things do you want to cluster? Then fully online as answer to the question of how big do you want windows to get.

[51]EdNote: this should be enough right?

[52]EdNote: potentially unnecessary

[53]EdNote: very colloquial

[54]EdNote: We need to invent some way of formalizing this intuition

[55]EdNote: actually T-n_{time}

[56]EdNote: Iko:This isn't super formal either REEEEEEEEEE

As a reminder, we defined by $g(t)$ the clustering induced by our boundary points, where $g(t)$ is equal to the most common element in the interval $[b_i, b_{i+1})$, for $t \in [b_i, b_{i+1})$ [57]. Then finding the optimal boundary points is just finding the answer to EdN:57

$$b_1, \dots, b_{k_{cluster}} = \underset{b_1,\dots,b_{k_{cluster}}}{\arg\min} \quad E(b_1, \dots, b_k)$$

[58] [59] The naive solution to this is just trying every combination of points, and then evaluating E. However while this will certainly find the optimal $b_1, \dots, b_k$, however it grows exponentially in $k_{cluster}$ and is thus only viable for finding one or two boundary points at most. EdN:58 EdN:59

Thus for more complex problems, we must try to find a good approximation. [60] EdN:60

3.1.1 Finding Approximate Boundaries

At first glance, finding good boundaries is trivial, since most clustering algorithms such as k-means clustering or [61] implicitly define decision boundaries, which we would just need to make explicit. However these decision boundaries would lie [62] in the two-dimensional space of $(time, clustering)$. The boundary points that we are looking for are one dimensional points in time, thus some care has to be taken. EdN:61 EdN:62

[63] EdN:63

One primitive approach, was to find for every clustering $i \in [1, \dots, n_{cluster}]$ the window of size w_s [64] that contains the maximum amount of labels [65] c_i corresponding to this clustering. Then we simply use the start of this window as a boundary point. [66] boundaries=[] For clustering in $[1, \dots, n_{cluster}]$: clusteringcount=0 bestpoint=0 for point in $[1, \dots, len(list) - windowsize]$: if [point,point+windowsize].count(clustering)¿clusteringcount: clusteringcount=[point,point+windowize].count(c bestpoint=point boundaries.append(point) return boundaries EdN:64 EdN:65 EdN:66

[67] EdN:67

This actually works surprisingly well, however it does need the extra parameter windowsize, and it doesn't work super well with data that has many different windowsizes, and has ugly combinations. Say $[1, 1, 2, 1, 2, 2]$ and a windowsize of 3, would have boundaries at 0, and at 2. Also has issues with the starting boundaries. Also it isn't theoretically satisfying.

[57]EdNote: lko: Is this phrased nicely? Should be just that it's actually in that interval

[58]EdNote: check if the prime shows well

[59]EdNote: I forgot, do you do a hanging . here?

[60]EdNote: Probably want to add a figure of what clusterings can look like. Something like 2 by 2 with different data. Make sure to include messy shit

[61]EdNote: support vector machine. Not really a method of clustering, but a way of finding a boundary

[62]EdNote: is this the spelling?

[63]EdNote: We formalized what the list would like above. Check that everything is consistent

[64]EdNote: terrible variable name. Look for better one later

[65]EdNote: check if we called them labels earlier. I really need to get this stuff consistent

[66]EdNote: pseudo code here. Make sure it's nice.

[67]EdNote: end pseudocode

[68] While the previous approach worked relatively well for "'nice"' [69] clusterings, with similar, known windowsizes and not much confusion, we also need an approach that works when the data gets more ugly. EdN:68 EdN:69

The first clustering algorithm that almost everyone learns [70] is k-means clustering [71]. It would seem natural at first glance to use it to find the decision boundaries as well. However, it is made more complex by the fact that the data points that we would be applying to it to would be from $[1, \ldots, n_{cluster}] \times [1, \ldots, T]$, i.e. the points that we would want to cluster consist of a pair of clustering and point in time. However, the distance for the clusterings can't just be the standard distance function, since a clustering label of 1 is in principle the same distance from a clustering label of 2 as it is from a clustering label 10. EdN:70 EdN:71

Thus we define the distance function $d^* : (\mathbb{N} \times \mathbb{R})^2 \to \mathbb{R}$ [72] by EdN:72

$$d^*((c_1, t_1), (c_2, t_2)) = \mu * \delta_{c_1,c_2} + |t_1 - t_2|$$

where $\delta_{i,j}$ is the Kronecker-delta.

Thus we define the distance between two points to be the distance of their clustering and their temporal distance. We scale the distance of their clusterings by a factor $\mu > 0$, since $\delta_{i,j}$ is always either 1 or 0, while t_i could range as high as 10000 in some testcases.

We choose a μ of about half the expected window size $w_{expected}$, as this will ensure that for some cluster with center c^*, every point in the interval $[c^* - w_{expected}/2, c^* + w_{expected}/2]$ with the same clustering as c^* will be assigned to the c^*'s cluster. [73] [74] Then the slightly modified $k-means$ we use is defined by [75] Initialize centers c_i^* Initialize clusterings $C_i$empty. While(not done) for clustering,time in clusterlist: EdN:73 EdN:74 EdN:75

$i = argmind^*((clustering, time), c_i^*)C_i.append((clustering, time)ForallC_i : c_i^* = (mostcommon(C_I[0]), mean(C_i[1])repeattillhappy$ [76] We note that this further departs from the conventional k-means algorithm by choosing not the average clustering as new center of a cluster, but instead the most common element. This is again a natural consequence of the nature of the clusterings as only integers. [77] EdN:76 EdN:77

After applying this technique, we are left with a list of clusters C_i consisting of points $(oldclusteringlabel, time)$ [78] and their accompanying center (c_i^*, t_i^*). To transform these into boundary points b_i^* [79], we simply project onto the EdN:78 EdN:79

[68]EdNote: obviously rephrase this.

[69]EdNote: check if these sort of quotes work like I want them to

[70]EdNote: I'm not sure this is a good transition

[71]EdNote: If I wnated to jsut cite this instead of explaining it, here would be a place to do so

[72]EdNote: check if this is properly formatted

[73]EdNote: More analysis can be done here on why this would be a cool choice and what the consequences would be, potentially some conditions as well

[74]EdNote: Still not entirely sure where/whether I should define k-means clustering

[75]EdNote: again pseudo code here

[76]EdNote: end pseudo code

[77]EdNote: Needs to be rephrased. clustering versus cluster is awkward, I need to find a good name to call things

[78]EdNote: check if there are proper spaces and such in there

[79]EdNote: check if this is how we called boundary points previously

temporal axis and are left with points t_i^*.

However these points correspond to centers of clusters, not decision boundaries. Naively, we could just take the middle point between two centers t_i as decision boundary, however this does not deal well with situations such as $[1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2]$. Intuitively, the best centers for two clusters would be $(1, 2)$ and $(2, 8)$. However if we then took the center, we would get a decision boundary $(8 + 2)/2 = 5$, instead of the clearly superior 4.

[80] Thus we must find a more suitable way to get boundaries from centers than just taking their middle points respectively. [81] We recall, that we not only have the centers (c_i^*, t_i^*), but also the clusters C_i available. The problem arises when we try to find the boundary between two clusters of different size, thus it stands to reason that we can use the sizes of our clusters to better approxmimate a good boundary by using the formula: EdN:80 EdN:81

$$b_i = \frac{|C_i|t_{i+1}^* + |C_{i+1}t^*i}{|C_i| + |C_{i+1}|}$$

If we apply this to our previous example, we get $b_i = \frac{4*2+8*5}{12} = 4$, which is the result we wanted.

Thus, the second algorithm for finding boundaries is given by [82] $Givenlist[c_1, \ldots, c_T]ofclusteringlabelsC_i, c_i$ EdN:82
$modifiedkmeans(c_1, ldots, c_T)Foriinrange(len(C_i)-1) : b_i = \frac{|C_i|*t^*i+1+|C_{i+1}|*t^*i}{|C_i|+|C_{i+1}|}$
[83] [84] EdN:83 EdN:84

3.1.2 Number of Clusters

Now that we have determined how to find boundaries, we must ask how many clusters are we looking for? Of course, if the number of desired clusters is known, we can simply look for the appropriate number of clusters.

However, if the appropriate number of clusters is unknown, we have to make some guess as to how many clusters to look for is appropriate [85]. EdN:85

To answer this question we look at diagram [86] . As we can see , for a too low choice of boundaries, certain boundaries are missed completely, such as the one by 1700. EdN:86

If we choose a too high number of clusters, some of the boundaries seem to all correspond to the same boundary,as seen for the boundary by 650 in the diagram, while some potentially finer clustering is achieved in different areas of the diagram. For the sample analyzed in the diagram, the first 250 timesteps consisted of a person walking, the different segmentations of the first real segment correspond to the person changing direction.

[80]EdNote: is this good paragraph wise
[81]EdNote: I don't like middlepoints, there should be a better word available
[82]EdNote: start pseudocode
[83]EdNote: end pseudocode
[84]EdNote: Might be appropriate to put a small subsection in with diagram taht shows how the two clustering algorithms perform on different data
[85]EdNote: are appropriate? Not sure onthis one
[86]EdNote: add in diagram here, that shows high amount of clusters versus low amount of clusters

Thus we conclude that if we are unsure on the amount of clusters/boundaries to look for, we should err on the side of caution and rather choose a too high number of boundaries than a too low one.

However we note that regardless of how many boundaries we choose to look for, the boundary at 800 [87] is not detected by the algorithm. One potential cause of this is that similarities to off-diagonal elements in the similarity matrix are detected, that might interfere with the correct choice of boundary. EdN:87

Hence we look at a possibly different choice of similarity matrix, to help alleviate the issue.

3.1.3 Choice of Similarity matrix

As discussed previously, the major choice in similarity matrix is choosing between the dtw-distance and euclidean distance, and further deciding how many off-diagonal elements to choose.

In diagram [88], we show five different choices of similarity matrix. The first choice is just using a full matrix filled with similarities computed using the euclidean distance, while the second is a full matrix of similarities computed using the dtw distance [89]. The last three matrices include EdN:88 EdN:89

[90] EdN:90

3.1.4 Heuristics for better clustering

[91] EdN:91

3.2 Mini Batch

Take fixed window sizes, find one boundary, work from there.

3.3 Full Online

[92] Essentially minibatch, but happy once a certain quality is reached. EdN:92

3.4 Parameters

Reducenumber, number of features used. Delta and amount of features used for distance matrix. Distance measure (dtw versus just standard versus lazy as fuck :D)

[87]EdNote: is x even accepted?
[88]EdNote: reference here
[89]EdNote: check if dtw was capitalized before
[90]EdNote: include diagram of similarity matrices
[91]EdNote: Call this Batch SFA cutting or something, just find some nice name for the technique
[92]EdNote: possibly not gonna happen, remove subsection later if necessary

4 Results

Compare with results from other papers. [93] EdN:93

4.1 Toydata

4.2 Real data

We use CMU motion capture data set.

4.3 SFA as preprocessing

4.4 Selfsimilarity with no SFA

[93]EdNote: First we want the direct comparisons, then just SFA as preprocessing for other methods, then potentially without any SFA at all

5 Conclusion

Idk I don't think it works super well right now.

References