Online slow feature analysis for the segmentation of motion data

Lukas Kohlhase

October 17, 2016

Abstract

In this paper we investigate an algorithm developed by Kompella et. al in [KLS12] for online slow feature analysis. We apply it to motion data to try to segment the data into meaningful chunks. We show that while we couldn't segment the data entirely into meaningful chunks, the method works in principle to identify key events, thus with some improvement using online Slow Feature Analysis for motion data segmentation could work.

1 Introduction

Time-series Segmentation is the process of splitting a time-series of data into several smaller segments according to some underlying properties of the time-series. This has many useful applications, for example in analyzing security footage, splitting a soundtrack into segments spoken by different people, or traffic pattern analysis. Several methods have been developed to automatically segment time series data, we base our broad classification on the one found in [KCHP93]

One broad category of techniques for segmentation are sliding window algorithms, these function by growing some window, until it exceeds a previously chosen error bound. Then another window is opened and the process is repeated until the end of the time series is reached. One major advantage of these methods is that they immediately lend themselves to online use, which is very useful for real life applications. However, they give poor results under some circumstances, as outlined in [SZ96].

Top-down approaches recursively segment the data into finer and finer segments until an error bound is reached. Similarly, but in the opposite direction, bottom down approaches, start with a segmentation of one data point per segment, that are continuously merged until a stopping requirement is reached. One of the disadvantages of these algorithms is that they don't lend themselves well to online use.

Because segmenting data as it comes in is very useful, there is evidently a need for online segmentation of data without the possible issues of conventional moving window approaches.

2 Online Slow Feature Analysis

The approach we will use in this paper is based on Slow Feature Analysis (SFA) and is outlined in [KLS12]. SFA is an unsupervised learning algorithm to extract slowly changing features from quickly changing input data. The technique mainly consists of first using Principal Component Analysis (PCA) to whiten the input data and then making temporal difference vectors from the whitened data points. Finally Minor Component Analysis (MCA) is used to extract the slow features, vectors that can be multiplied with the original data points to get scalar values that are used for classification. For both MCA and PCA, online implementations exist, thus it is possible to use SFA in an online fashion.

2.1 Incremental PCA

The online version of PCA we will be using is called Candid Covariance-free Incremental Principal Component Analysis (CCIPCA) [WZH03].

Algorithm 1 CCIPCA

```
Compute the first $k$ dominant eigenvectors, $v_1(n), v_2(n), \ldots, v_k(n)$ directly from data $u(1), u(2), \ldots$
For $n = 1, 2, \ldots$:
    1. $u_1(n) = u(n)$
    2. For $i = 1, 2, \ldots, min\{k, n\}$:
        i If $i = n$, initialize the $i$th eigenvector as $v_i(n) = u_i(n)$
        ii Else
            $v_i(n) = (1-\eta)v_i(n-1) + \eta u_i(n) u_i^T(n) \frac{v_i(n-1)}{||v_i(n-1)||}$
            $u_{i+1}(n) = u_i(n) - u_i^T(n) \frac{v_i(n)}{||v_i(n)||} \frac{v_i(n)}{||v_i(n)||}$
```

The eigenvalues λ_i associated to the eigenvectors v_i are estimated as $\lambda_i = ||v_i||$. The computed eigenvectors $v_i(t)$ converge to the correct eigenvectors, however it is notable that the lower order eigenvectors converge a lot faster than, the higher order ones. Thus, it CCIPCA is not suitable for online MCA, as for that we need exactly the higher order principal components.

2.2 Online MCA

The algorithm presented here is based on [KLS12] and provides the first J minor components. Since MCA will be used on temporal difference vectors of the whitened data, we will use $z(t)$ for these temporal difference vectors. This algorithm converges to the correct minor component, assuming a learning rate η_2 is chosen that is small enough such that $\eta_2 \lambda_1 < 0.5$ and that the initial estimate of the minor component $w_1(0)$ is chosen such that $||w_1(0)||^2 \leq \frac{1}{2\eta_2}$ and $w_1(0)$ is not orthogonal to the true minor component w^*. To estimate the higher order minor components, a factor γ needs to be chosen that is larger than the largest eigenvalue of $E[z(t)z^T]$, which can be estimated as $\lambda_1 + \epsilon$ for some small ϵ.

Algorithm 2 Online MCA

```
Choose $\gamma(t)$ such that $\gamma > E[z(t)z^T(t)]$
For $t = 1, \dots,$
  1. $C_1(t) = z(t)z(t)^T$
  2. For $i = 2, \dots, J$
       i  $w_i(t) = 1.5 * w_i(t-1) - \eta_2 C_i w_i(t-1) - \eta_2 (w_i^T(t-1) w_i(t-1)) w_i(t-1)$
       ii $C_i(t) = C_{i-1}(t) + \gamma(t)(w_{i-1}(t) w_{i-1}^T(t)) / (w_{i-1}^T(t) w_{i-1}(t))$
```

We note that C_i are $n \times n$ square matrices.

2.3 Putting the pieces together

Now all that is left is adding these pieces together to achieve a full algorithm. We modify the algorithm slightly from the version presented in [KLS12], by assuming that the preprocessed data has 0 mean. We choose a whitening dimension of k and extract J slow features.

We can clearly see that the most computationally expensive parts of the algorithm are either the online PCA part or the online MCA part. We can see that for CCIPCA, the main operations are additions of vectors and dot products of vectors $u_i^T * v_i(n)$, which are performed $t * k$ times and each cost m, the dimension of input data, time. Hence the time complexity of $CCIPCA$ is inside $O(t * k * m)$. For online MCA, the main costs are matrix multiplications with vectors with costs of k^2, matrix additions with costs of k^2, and vector multiplications to form matrices $w_{i-1}^T w_{i-1}$, with costs of k^2. Thus the complexity of the online MCA algorithm is inside $O(t * k^2 * J)$.

Evidently, to reduce the total cost, the most effective way is to choose a low whitening dimension k, corresponding to whitening with only the first k principal components. This would decrease the time complexity of the PCA part linearly by k and quadratically decrease the time complexity of the MCA part. Additionally, it should not reduce the quality of the results by a large

Algorithm 3 Online SFA

```
For $t = 0, 1, \ldots$
    1. **Sense:** Receive input vector $u(t)$
    2. **CCIPCA:** Apply Algorithm 1 to find the first $k$ principal components
    3. **Whitening:** Let $V(t)$ contain the normed estimates of the $k$ principal components, and create the $k \times k$ diagonal matrix $D(t)$, where $D_{i,i} = 1/\sqrt{\lambda_i(t)}$. Then $z^*(t) = V(t)D(t)u(t)$
    4. **Derivative Signal:** Set $z(t) = z^*(t) - z^*(t-1)$
    5. **Compute $\lambda$:** Use Algorithm 1 to find the first principal component of $z$ and it's associated eigenvalue $\lambda_1$ to set $\gamma > \lambda_1$
    6. **Online MCA:** Update estimates of the $J$ least significant components according to Algorithm 2
    7. **Output:** Let $W(t)$ contain the current estimates of the slow features. Then $y(t) = z^T(t)W(t) = u^T(t)D^T(t)V^T(t)W(t)$ is the SFA output
```

margin, since the minor principal components determined by CCIPCA only converge very slowly, and thus hold not much more information than random vectors would. Thus, if we choose a low enough k, the time complexity of the entire algorithm is dominated by CCIPCA and is thus in $O(t * k * m)$, which means we can do almost arbitrary preprocessing by expanding the input space, since an increase in input size only reflects linearly in time complexity.

One of the normal avenues for further improvement would be to use kernels to replace costly dot products of vectors. However the only dot products are of the form $u_i^T * v_i$, so input vectors multiplied with principal components. If we were to use a kernel, we would have a map $K(u_i, w_i) = \Phi(u_i)*\Phi(w_i) = \Phi(u_i)*v_i$, from some preprocessing Φ. However CCIPCA finds v_i directly, and since Φ is not invertible, we cannot find $w_i = \Phi^{-1}$ for use in any kernels.

3 Application to data

The dataset we used for our experiment was a subset of the CMU motion capture database, available at [dat]. The dataset consists of 62-dimensional vectors, containing the root position and joint angles for the depicted person. Additionally it is presegemented into several parts such as ”walk”, ”punch”, or ”swing arms”.

We played around with several methods of data expansion, such as just using the linear data, polynomial expansion of several degrees, and applying various trigonometric functions to the data points. The expansion that worked best was using polynomial expansion with degree two. Higher degrees of expansion had very similar results, but ended up taking a lot more time.

We used a η of 0.005 for both the MCA and CCIPCA parts of the algorithm, as setting it lower led to nonsensical results that were too dependent on the random initialization. Since we used a polynomial data expansion of degree, the input vectors were 2015-dimensional. For much of our analysis, we centered the data beforehand instead of using the fully online approach of using a moving updated average. This lead to less spiky and more nice looking slow features, but left them essentially unchanged. Finally, we used a whitening dimension of 20 and also 20 different slow features.

Because making graphs look nice is a lot of work and took a lot of trial and error, all of the figures shown were generated from scratch many times and never changed drastically, despite random initializations, the relevant points talked about later never changed.

3.1 Data set 1, rotating arms

The first dataset we examine is dataset 11 use by [ZDH13] and available at [dat]. It features a person first walking in a rough circle, then swinging their arms, and then walking again. The first 3 features extracted can be seen in Figure 1.

At first glance we can see that each of the curves has a macro shape, the red one goes from relatively constant around $y = 8$ to some variation around $y = -3$ until moving further down and then back up again, and a more local sine wave like shape. After watching the relevant motion data several times, I could not detect any meaning in the macro shape of any of the three pictures curves. It does not seem to coincide with the movement of the person, since their root position does not change during the first right arms rotation section, and it does not reflect the direction that the person is walking, since both walking movements follow the same trajectory and start from the same position.

However, the local sine wave shape corresponds very well to the position of the arms. If we look in particular at the red curve, it's local minima are at the same frames that the person in the animation has one or both arms at their back. Since all three curves have very similar local extrema, this seems to be true for the other slow features as well, though they might be measuring a slightly different arm position at their extrema. This would also explain the somewhat erratic nature of the curves during the two walking segments, since the figure is swinging their arms while walking. However I can't explain why the second walk and both arms rotations are so different from the first ones, at least visually they look very similar in the animation.

In Figure 2, we handled the same dataset, except with the original expanded data, without centering it beforehand. The same conclusions still hold, the macro shape of the curves does not seem to hold any information, while the same local peaks are still present, although not as pronounced and not as nice looking.

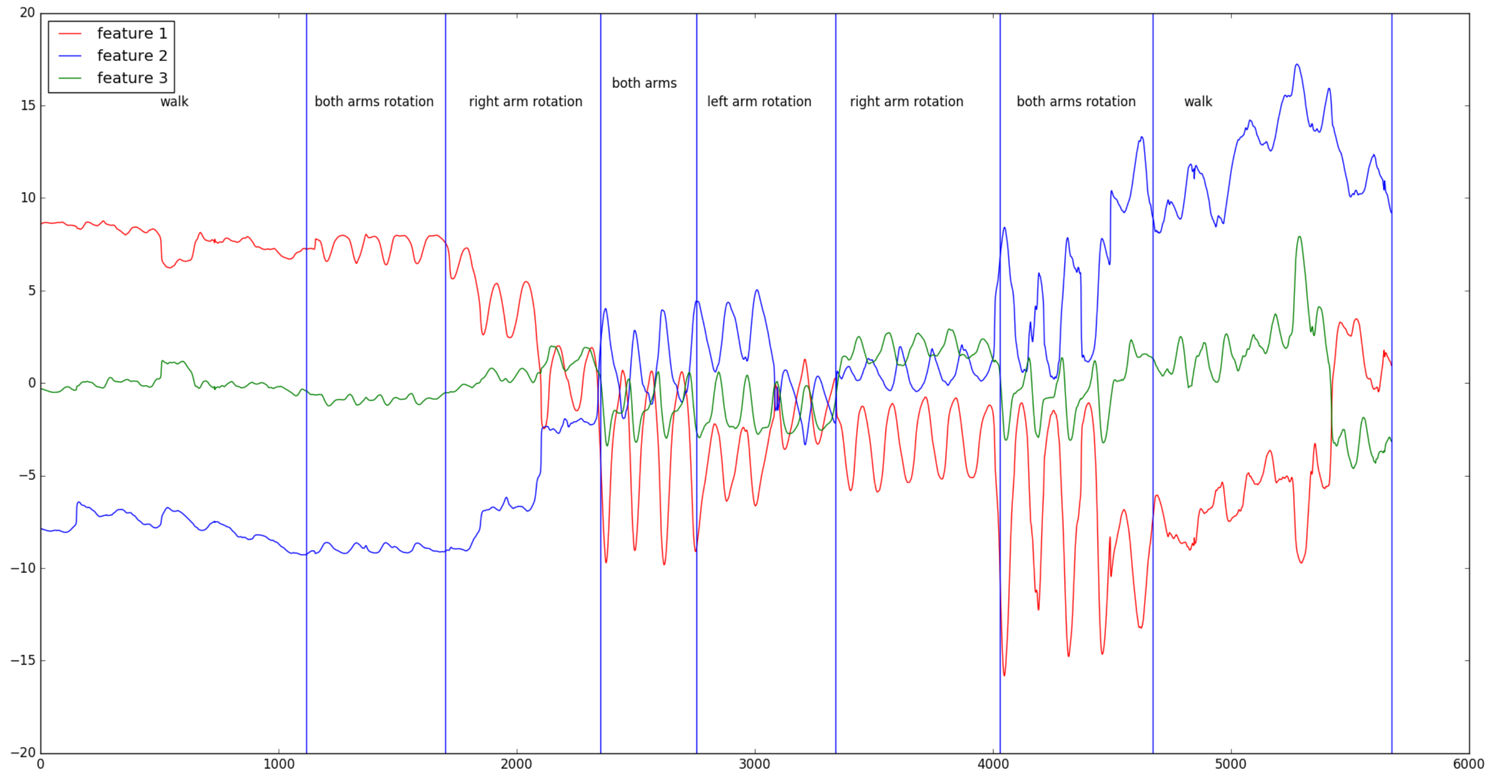

Figure 1: The three slowest feature for the first dataset

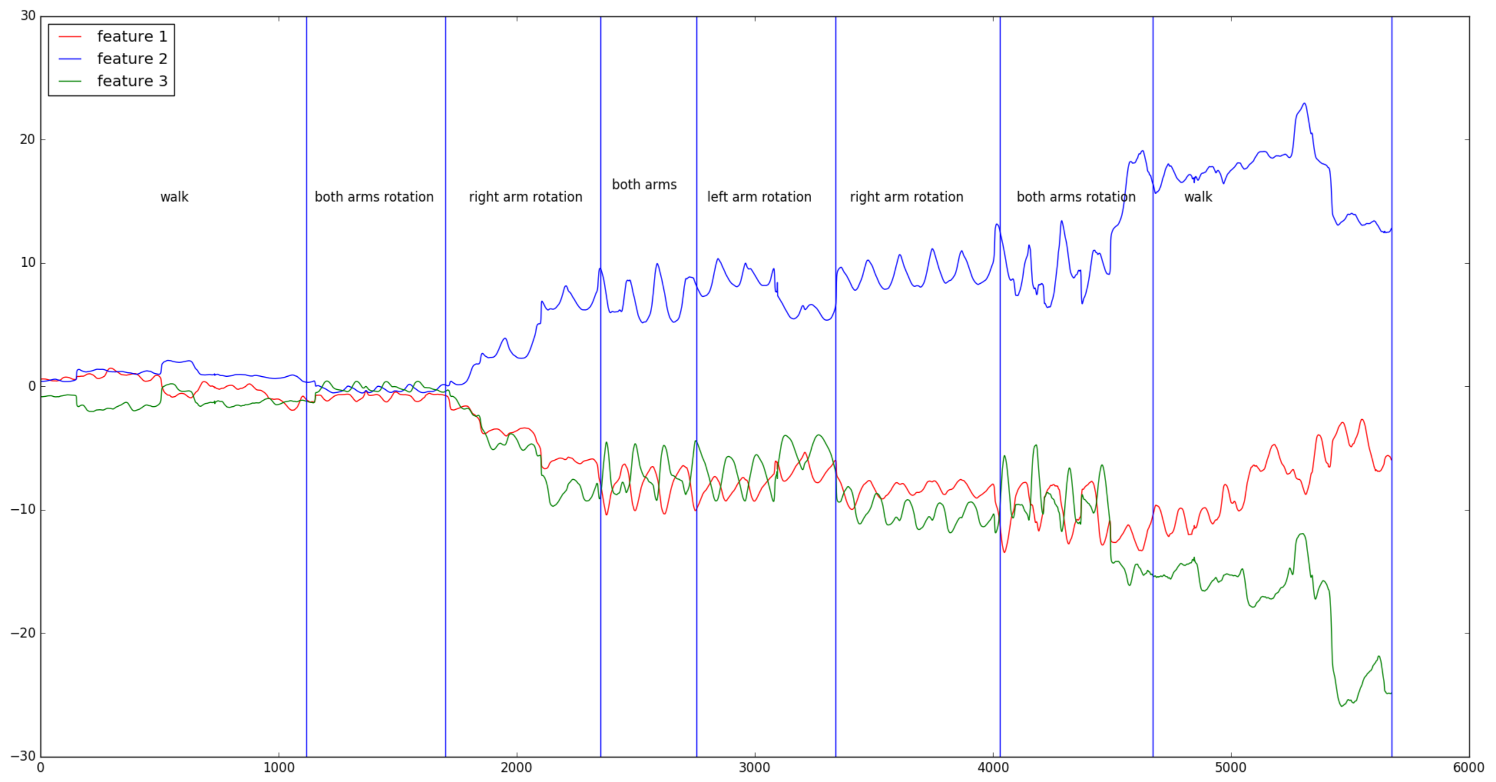

Figure 2: The three slowest feature for the first dataset, using the moving average approach

3.2 Data set 2, punching and kicking

In the second dataset, the person does not rotate their arms, but instead does punches, kicks and some walking. The three slowest features are represented in Figure 3.

The results are similar to the previously seen situation, in that the macro structure does not seem to have any relation to the segments of kicking, punching or walking. An additional similarity to before is that I can't make out any micro structure for the walking at all. However, it is possible to identify the exact points at which the person picked from the noticeable spikes that the first feature exhibits. These correspond to the points of maximum extension of the kick, when it would have hit a target. In contrast to the first slow features, the slowest feature does not seem to hold any information about the punching motions. However the second feature does. During the punching segments, local maxima align with the pictured person having both arms at their sides, while local minima align with the full extension of the arm during a punching motion.

4 Conclusion

We did not manage to use online Slow Feature Analysis to segment the provided datasets in the same manner as they were segmented by the creators of the dataset. The output did not seem to have any identifying characteristics on a macro level, such as the absolute value of the slow feature, that could be used to segment them. However, on a local level there was a marked difference between the different movement types, rotation of the arms led to a shape that looked locally much like a sine curve, while walking led locally to a shape looking more like the price of stocks.

A potential improvement would be to detect this change in local structure to also be able to segment the dataset similarly to how it was originally segmented.

Finally, for both of the datasets we looked at in detail we were able to identify individual actions,such as when arms were behind the person's back during arm rotation, for kicking, punching, and rotating arms. The unifying characteristic between these actions was that the person doing them was relatively stationary at the time. This might warrant further investigation.

To sum it all up, we did not quite manage to achieve the segmentation we wanted to, however we were able to apply it to parts of the dataset, which shows that with further improvements this method could prove viable.

References

[dat] Temporal clustering of human behaviour. http://www.f-zhou.com/tc.html. Accessed: 2016-10-15.

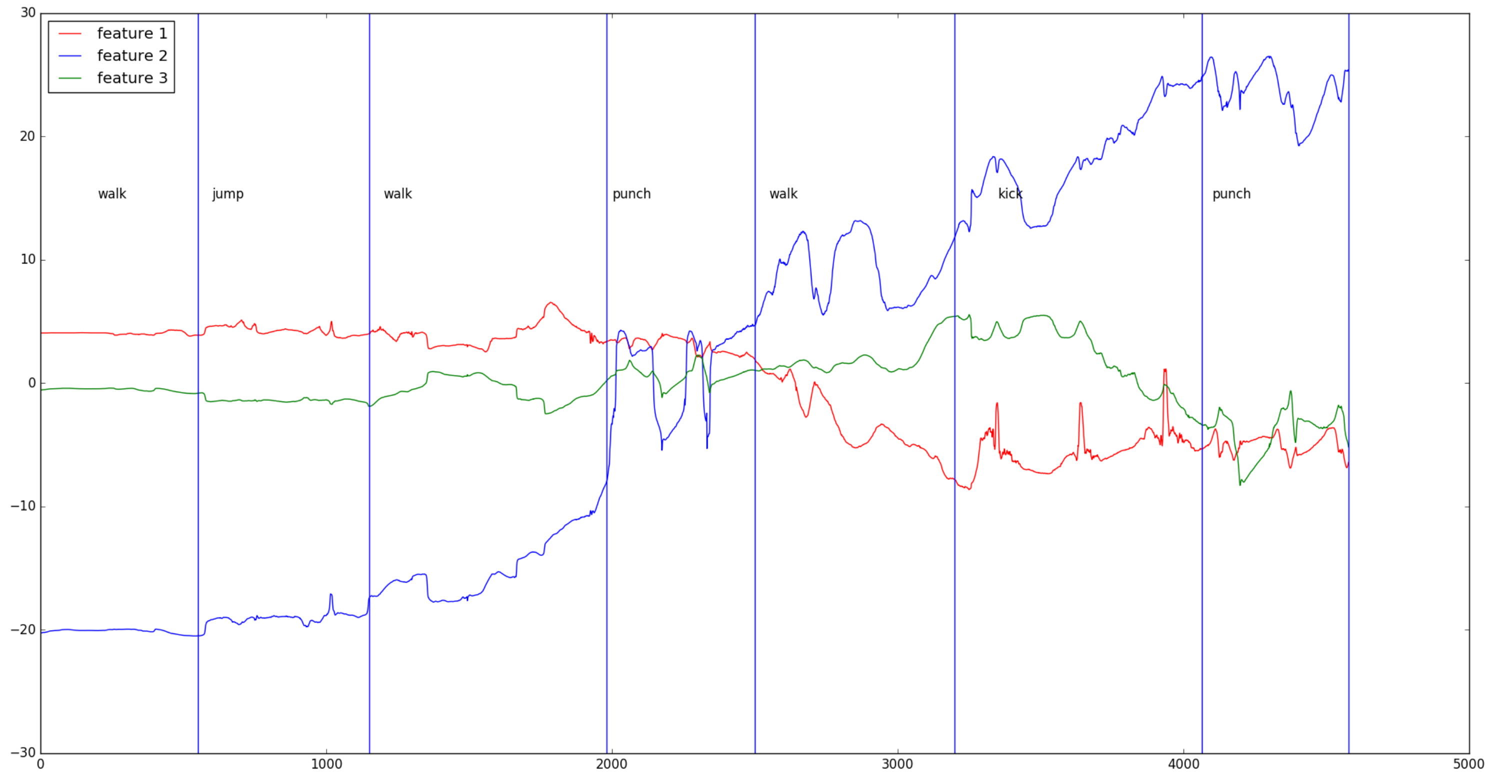

Figure 3: The three slowest feature for the second dataset

[KCHP93] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. In *In an Edited Volume, Data mining in Time Series Databases. Published by World Scientific*, pages 1–22. Publishing Company, 1993.

[KLS12] Varun Raj Kompella, Matthew Luciw, and Jürgen Schmidhuber. Incremental slow feature analysis: Adaptive low-complexity slow feature updating from high-dimensional input streams. *Neural Comput.*, 24(11):2994–3024, November 2012.

[SZ96] Hagit Shatkay and Stanley B. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*, ICDE '96, pages 536–545, Washington, DC, USA, 1996. IEEE Computer Society.

[WZH03] Juyang Weng, Yilu Zhang, and Wey-Shiuan Hwang. Candid covariance-free incremental principal component analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(8):1034–1040, August 2003.

[ZDH13] Feng Zhou, Fernando De la Torre Frade, and Jessica K Hodgins . Hierarchical aligned cluster analysis for temporal clustering of human motion. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 35(3):582–596, March 2013.