AutoSplit: Fast and Scalable Discovery of Hidden Variables in Stream and Multimedia Databases*

Jia-Yu Pan[1], Hiroyuki Kitagawa[2], Christos Faloutsos[1], and Masafumi Hamamoto[2]

[1] Carnegie Mellon University, Pittsburgh PA 15213, USA
[2] University of Tsukuba, Tennohdai, Tsukuba, Ibaraki 305-8573, Japan

Abstract. For discovering hidden (latent) variables in real-world, non-gaussian data streams or an n-dimensional cloud of data points, SVD suffers from its orthogonality constraint. Our proposed method, "AutoSplit", finds features which are mutually independent and is able to discover non-orthogonal features. Thus, AutoSplit (a) finds more meaningful hidden variables and features, (b) it can easily lead to clustering and segmentation, (c) it surprisingly scales linearly with the database size and (d) it can also operate in on-line, single-pass mode. We also propose "Clustering-AutoSplit", which extends the feature discovery to multiple feature/bases sets, and leads to clean clustering. Experiments on multiple, real-world data sets show that our method meets all the properties above, outperforming the state-of-the-art SVD.

1 Introduction and related work

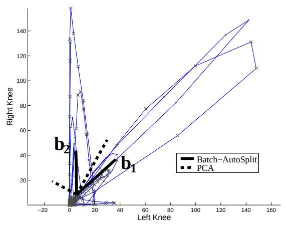
In this paper, we focus on discovering patterns in (multiple) data streams like stock-price streams and continuous sensor measurement, and multimedia objects such as images in a video stream. Discovery of the essential patterns in data streams is useful in this area, for it could lead to good compression, segmentation, and prediction. We shall put the related work into two groups: dimensionality reduction and streaming data processing.

Dimensionality reduction/feature extraction Given a cloud of n points, each with m attributes, we would like to represent the data with fewer attributes/features but still retain most of the information. The standard way of doing this dimensionality reduction is through SVD (Singular Value Decomposition). SVD finds the best set of axes to project the cloud of points, so that the sum of squares of the projection errors is minimized (Figure 1(a)). SVD has

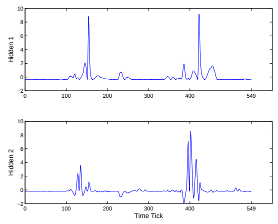
* Supported in part by Japan-U.S. Cooperative Science Program of JSPS; grants from JSPS and MEXT (#15017207, #15300027); the NSF No. IRI-9817496, IIS-9988876, IIS-0113089, IIS-0209107, IIS-0205224, INT-0318547, SENSOR-0329549, EF-0331657; the Pennsylvania Infrastructure Technology Alliance No. 22-901-0001; DARPA No. N66001-00-1-8936; and donations from Intel and Northrop-Grumman.

been used in multiple settings: for text retrieval [1], under the name of Latent Semantic Indexing (LSI); for face matching in the eigenface project [2]; for pattern analysis under the name of Karhunen-Loeve transform [3] and PCA [4]; for rule discovery [5]; and recently for streams [6] and online applications [7]. Recently, approximate answers of SVD for timely response to online applications have also been proposed [8–11].

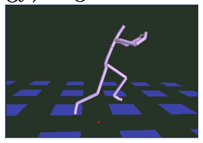
Streaming data processing Finding hidden variables is useful in time series indexing and mining [12], modeling [13, 14], forecasting [15] and similarity search [16, 17]. Fast, approximate indexing methods [18, 19] have attracted much attention recently. Automatic discovery of "hidden variables" in, for example, object motions would enable much better extrapolations, and help the human analysts understand the motion patterns.



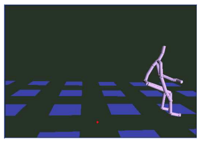
(a) (Energy) Right knee vs left knee



(b) Hidden variables $\mathbf{h_1}$ and $\mathbf{h_2}$



(c) Take off



(d) Landing

Fig. 1. (AutoSplit versus SVD/PCA: "Broad jumps") (a): the right knee energy versus left knee energy during the jumps: take off (c) and landing (d). Two jumps are performed at time ticks 100 and 380. In (a), (Batch-)AutoSplit vectors $\mathbf{b_1}$, slope 1:1, corresponds to "landing"; $\mathbf{b_2}$, slope -1:60, for "take off". (b) the hidden variables $\mathbf{h_1}$ (top) and $\mathbf{h_2}$ (bottom) of $\mathbf{b_1}$ and $\mathbf{b_2}$, respectively.

Although popularly used, SVD suffers from its orthogonality requirement for real world data whose distribution is not gaussian. For example, in Figure 1(a), SVD proposes the two *dash* orthogonal vectors as its basis vectors, while completely missing the "natural" ones ($\mathbf{b_1}$, $\mathbf{b_2}$). Is there a way to automatically find the basis vectors $\mathbf{b_1}$ and $\mathbf{b_2}$? Generally, we would like to have a method which (a) finds meaningful feature vectors and hidden variables (better coincide with the true unknown variables which generate the observed data streams), (b) can work in an unsupervised fashion, (c) scales linearly with the database size, (d) is able to operate in on-line, single-pass mode (to cope with continuous, unlimited data streams). Experiments on multiple, real data sets, from diverse settings (motion capture data for computer animation, stock prices, video frames) show that the proposed AutoSplit method and its variants achieve the above properties.

2 Proposed Method

There are two major concepts behind AutoSplit: the *basis vectors*, and the *hidden variables*. The basis vectors are the analog of the eigenvectors of SVD, while the hidden variables are the sources/variables controlling the composition weights of these basis vectors when generating the observed data. We use the "broad jumps" data set in section 1 to illustrate these concepts.

Basis vectors and hidden variables The "broad jumps" data set (Figure 1) is a motion capture data set from [20]. The actor performed two broad jumps during the recording period. Our data set is a n-by-m data matrix $\mathbf{X}$=$[x_{i,j}]$ with n=550 rows (time-ticks) and m=2 columns (left and right knee energy). Figure 1(a) shows the scatter plot of the data points: $x_{i,1}$ versus $x_{i,2}$, or, informally: right-knee(i) versus left-knee(i), for time ticks $i = 1, \ldots, n$. For visualization purposes only, data points at successive time-ticks are connected with lines - neither SVD nor AutoSplit use this sequencing information.

For the "broad jumps" data, we would like to (a) *discover the structures of the action* (take-off, landing), and (b) *partitions the action sequences into homogeneous segments.* Notice that the majority of points are close to the origin, corresponding to the time when the knees are not moving much (idle and "flying"). However, there are two pronounced directions, one along the 45^o degree line, and one almost vertical. As shown, SVD fails to spot either of the two pronounced directions. On the other hand, AutoSplit clearly locks on to the two "interesting" directions, unsupervisedly.

(Observation 1) *Playing the animation and keeping track of the frame-numbers (= time-ticks), we found that the points along the 45^o degree line ($\mathbf{b_1}$) are from the landing stage of the action (2 knees exert equal energy, Figure 1(d)), while the ones on the near-vertical AutoSplit basis ($\mathbf{b_2}$) are from the take-off stage (only right knee is used, Figure 1(c)).*

Exactly because AutoSplit finds "good" basis vectors, most of the data points lie along the captured major axes of activities. Thus, the encoding coefficients (hidden variables) of any data point are all closed to zero, except for the one which controls the axis on which the data point lies (Figure 1(b)). Inspecting the data found that $\mathbf{h_1}$ controls the landing, while $\mathbf{h_2}$ controls the take off. The distinct "fire-up" periods of the hidden variables could lead to clean segmentation.

2.1 AutoSplit: Definitions and discussion

As shown in Figure 1(a), the failure of SVD partly comes from the orthogonal constraint on its basis vectors. Since our goal is to find clean features, therefore, instead of constraining on the orthogonality, we search for basis vectors which maximize the mutual independence among the hidden variables. The idea is that *independence leads to un-correlation, which implies clean, non-mixed features.*

(Definition 1: Batch-AutoSplit) *Let* $\mathbf{X}_{[\mathbf{nxm}]}$ *be the* ***data matrix*** *representing n data points (rows) with m attributes (columns). We decomposes* $\mathbf{X}_{[\mathbf{nxm}]}$ *as a linear mixture of l bases (rows of the* ***basis matrix*** $\mathbf{B}_{[\mathbf{lxm}]}$*), with weights in the columns of the* ***hidden matrix*** $\mathbf{H}_{[\mathbf{nxl}]}$ *(l: the number of hidden variables,* $l \leq m$*).*

$$\mathbf{X}_{[\mathbf{nxm}]} = \mathbf{H}_{[\mathbf{nxl}]}\mathbf{B}_{[\mathbf{lxm}]},$$

$$\begin{bmatrix} -\boldsymbol{x_1}- \\ -\boldsymbol{x_2}- \\ \vdots \\ -\boldsymbol{x_n}- \end{bmatrix} = \begin{bmatrix} | & | & \cdots & | \\ \boldsymbol{h_1} & \boldsymbol{h_2} & \cdots & \boldsymbol{h_l} \\ | & | & \cdots & | \end{bmatrix} \begin{bmatrix} -\boldsymbol{b_1}- \\ -\boldsymbol{b_2}- \\ \vdots \\ -\boldsymbol{b_l}- \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & \dots & h_{1l} \\ h_{21} & h_{22} & \dots & h_{2l} \\ \vdots & \vdots & & \vdots \\ h_{n1} & h_{n2} & \dots & h_{nl} \end{bmatrix} \begin{bmatrix} -\boldsymbol{b_1}- \\ -\boldsymbol{b_2}- \\ \vdots \\ -\boldsymbol{b_l}- \end{bmatrix} . \blacksquare$$

We model each data point $\mathbf{x_i}$ as a linear combination of the basis vectors $\mathbf{b_k}$ (features). The value $b_{k,j}$ indicates the weight of the j-th attribute for the k-th hidden variable, where hidden variables represent the unknown data generating factors (e.g., the economic events to a share price series). In the "broad jumps" example, there are n=550 data points $\mathbf{x_i}$, each is m=2 dimensional. The l=2 basis vectors were 2-d vectors, with values $\mathbf{b_1}$=$(15.51, 14.12)$ ($\sim 45^o$ degree line), and $\mathbf{b_2}$=$(-0.29, 17.65)$ (vertical line).

Figure 2 gives the outline of Batch-AutoSplit. The analysis behind Batch-AutoSplit is based on ICA (Independent Component Analysis) [21]. There is a large literature on ICA, with several alternative algorithms. The one most related to Batch-AutoSplit is [22].

Batch-AutoSplit assumes the number of hidden variables is the same as the number of attributes, i.e., $\mathbf{B}$ is a square matrix. The number of hidden variables, l, is controlled by the *whitening* step of the algorithm. The whitening of matrix $\mathbf{A}$ is obtained by first making $\mathbf{A}$ zero mean (and we get $\mathbf{A_0}$), and then apply SVD on $\mathbf{A_0}$=$\mathbf{U}\Lambda\mathbf{V^T}$. The whitening result is $\hat{\mathbf{A}}$=$\mathbf{U_l}$, where $\mathbf{U_l}$ is the first l columns of $\mathbf{U}$. Also, the Frobenius norm of the matrix $\mathbf{A}_{[\mathbf{nxm}]}$ is defined as $||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} a_{i,j}^2}$.

- Given: The data matrix $\mathbf{X}$, and a randomly initialized $\mathbf{B}$.
- Step 1: Whiten the data matrix $\mathbf{X}$, and get $\hat{\mathbf{X}}$. Initialize $\Delta\mathbf{B}$, such that $||\epsilon\Delta\mathbf{B}||_F > \delta$, where ϵ controls the size of each gradient step, and is usually reduced as more iterations are done, and δ is some pre-defined threshold.
- Step 2: While $||\epsilon\Delta\mathbf{B}||_F > \delta$.
  - Step 2.1: Compute $\mathbf{H}$=$\hat{\mathbf{X}}\mathbf{B}^{-1}$.
  - Step 2.2: Compute the gradient $\Delta\mathbf{B} = -\mathbf{B^T}\mathbf{Z^T}\mathbf{H} - n\mathbf{B^T}$, where $\mathbf{Z} = -\mathrm{sign}(\mathbf{H})$.
  - Step 2.3: Update $\mathbf{B}$=$\mathbf{B}+\epsilon\Delta\mathbf{B}$.

Fig. 2. Batch-AutoSplit algorithm: $\mathbf{B_{out}}$=Batch-AutoSplit($\mathbf{X}$,$\mathbf{B}$)

2.2 Proposed method: AutoSplit and Clustering-AutoSplit

Figure 3 extends the basic Batch-AutoSplit algorithm to process online data streams. AutoSplit takes into a infinite stream of data items (grouped into windows $\mathbf{X_0}$,$\mathbf{X_0}$,...) , and continuously output the estimated bases $\mathbf{B_0}$,$\mathbf{B_1}$,.... The memory requirement of the AutoSplit algorithm is tunable, by setting the number of data items (n) to be processed at each loop iteration. The computation time at each update is *constant (O(1))*, given fixed number of data points at each update. In fact, the actual time for each update is tiny, for only a couple of small matrix multiplications and additions are required at each update. Note that AutoSplit is capable of adapting to gradual changes of hidden variables, which is desirable for long-term monitoring applications, where the underlying hidden variables may change over time.

- Given: A infinite stream of data items, every n items are grouped as a data batch $\mathbf{X_l}$, $(l = 1, 2, \dots)$. Data patches could be overlapped.
- Step 1: When $\mathbf{X_0}$ is available, initialize $\mathbf{B_0}$ randomly. Let $l = 0$.
- Step 2: While $\mathbf{X_{l+1}}$ is not available, do
  - $\mathbf{B_l}$ = Batch-AutoSplit($\mathbf{X_l}$, $\mathbf{B_l}$).
- Step 3: $\mathbf{B_{l+1}}$=$\mathbf{B_l}$. Goto Step 2.

Fig. 3. AutoSplit algorithm: ($\mathbf{B_0}$,$\mathbf{B_1}$,...) = AutoSplit($\mathbf{X_0}$,$\mathbf{X_1}$,...)

Real world data is not always generated from a single distribution, instead, they are from a mixture of distributions. Many studies model this mixture by a set of Gaussian distributions. Since real world data is often non-Gaussian, we propose "Clustering-AutoSplit", which fits data as a mixture of AutoSplit bases.

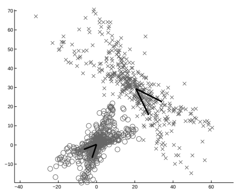


Fig. 4. Clustering-AutoSplit The number of clusters is set to $k = 2$. Each set of AutoSplit bases is centered at the mean of data points in a cluster.

Figure 4 shows an synthetic example of 2 clusters on the 2-dimensional plane. Each cluster is specified by a set of 2 bases. Note that by fitting the data into the mixture model, the data items are automatically clustered into different classes. Figure 5 gives the outline of the Clustering-AutoSplit algorithm. Note that we can also use AutoSplit algorithm, instead of Batch-AutoSplit, in Step 2.5 of the algorithm, yielding an online clustering algorithm.

- Given: k is the number of clusters to be found, and $\mathbf{X}$ has the data items.
- Step 1: Initialize $\mathbf{B_j}$'s and $\mathbf{c_j}$'s randomly, $j = 1, \ldots, k$. $\mathbf{B_j}$ and $\mathbf{c_j}$ are the bases and the mean of the j data cluster, respectively.
- Step 2: While the changes on $\mathbf{c_j}$'s remain large (above some threshold δ),
  - Step 2.1: For each data point $\mathbf{x}_i$, compute its likelihood of belonging to the j-th cluster,

$$f_{ij} = \frac{\prod_{r=1}^{l} f_h(h_{ir})}{|det(\mathbf{B}_j)|},$$

    where $\mathbf{x}_i = \sum_{r=1}^{l} h_{ir}\mathbf{b}_{jr}$, l is the number of bases for cluster j, and $f_h(h_{ir}) \propto exp(-|h_{ir}|)$.
  - Step 2.2: Compute the relative weight of x_i to cluster j, $p_{ij} = f_{ij}/(\sum_{k=1}^{n} f_{kj})$.
  - Step 2.3: Update $\mathbf{c}_j$'s: $\mathbf{c}_j = \sum_{i=1}^{n} p_{ij}\mathbf{x}_j$.
  - Step 2.4: Cluster each point $\mathbf{x}_i$ to the cluster of maximum likelihood. Let the k clusters be $\mathbf{C_1}, \ldots, \mathbf{C_k}$.
  - Step 2.5: For each cluster j, update $\mathbf{B_j}$=Batch-AutoSplit($\mathbf{C_j}$,$\mathbf{B_j}$) (Figure 2).

Fig. 5. Clustering-AutoSplit: ($\mathbf{B_1}$,$\mathbf{c_1}$, ... ,$\mathbf{B_k}$,$\mathbf{c_k}$)=Clustering-AutoSplit(k,$\mathbf{X}$)

3 Experimental Results

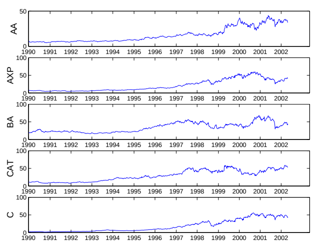
In this section, we show the experimental results of applying (a) AutoSplit to the real world share price sequences and (b) Clustering-AutoSplit to the video frames. We also empirically examine the quality and scalability of AutoSplit.

3.1 Share price sequences

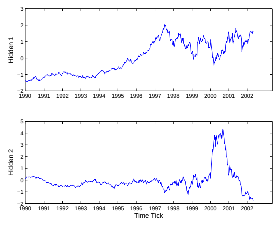
The share price data set (DJIA) contains the weekly closing prices of the m=29 companies in the Dow Jones Industrial Average, starting from the week of January 2, 1990 to that of August 5, 2002, and gives data at n=660 time ticks per company. Closing prices collected at the same week/time-tick are grouped into a 29-D vector, i.e., a company is an attribute. The resulting data matrix $\mathbf{X}$ is 660-by-29.

Before doing AutoSplit, we preprocess the data matrix $\mathbf{X}$ and make the value of each company/attribute zero-mean and unit-variance. To extract l=5 hidden variables, the dimensionality is first reduced to 5 from 29 using whitening (section 2.1). Figure 6(b) shows the 2 most influential hidden variables ($\mathbf{h_1}$, $\mathbf{h_2}$). We would like to understand *what do these hidden variables stand for*?

Table 1(a) lists the top 5 companies with largest and smallest contributions $b_{1,j}$ to the hidden variable $\mathbf{h_1}$ (top of Figure 6(b)), where index j is the index



(a) 5 members of DJIA



(b) Hidden variables of DJIA

Fig. 6. (Share closing price (DJIA, 1990-2002)) (a) AA: Alcoa, AXP: America Express, BA: Boeing, CAT: Caterpillar, C: CitiGroup. (b) (Top) Probably the general trend of share prices. (Bottom) Probably the Internet bubble.

$b_{1,j}$: Contribution to $\mathbf{h_1}$				$b_{2,j}$: Contribution to $\mathbf{h_2}$			
Highest		Lowest		Highest		Lowest	
CAT	0.938512	T	0.021885	INTC	0.641102	MO	-0.194843
BA	0.911120	WMT	0.624570	HWP	0.621159	IP	-0.089569
MMM	0.906542	INTC	0.638010	GE	0.509164	CAT	0.031678
KO	0.903858	HD	0.647774	AXP	0.504871	PG	0.109576
DD	0.900317	HWP	0.658768	DIS	0.490529	DD	0.133337
(a)				(b)			

Table 1. Company contributions according to the hidden variables: $\mathbf{h_1}$, $\mathbf{h_2}$. j is the index to companies. (INTC: Intel, AXP:American Express, DIS:Disnet, MO:Philip Morris, PG:Procter and Gamble, DD: Du Pont)

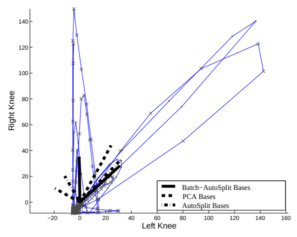
to the different companies. As shown, all companies have strong positive contributions (about 0.6 to 0.9) except AT&T (symbol: T). The regularity of contributions among all companies suggests that $\mathbf{h_1}$ represents the general trend of share price series.

On the other hand, the hidden variable $\mathbf{h_2}$ (bottom of Figure 6(b)) is mostly silent (near zero-value) except a sharp rise and drop in year 2000. This seems to correspond to the "Internet bubble". To verify this, we can check the companies' contributions $b_{2,j}$ on this hidden variable $\mathbf{h_2}$. Table 1(b) lists the companies having the 5 highest and 5 lowest contributions $b_{2,j}$ on $\mathbf{h_2}$. Companies having big contributions are mostly technical companies and service (financial, entertainment) providers, while those of near-zero contributions are bio-chemical and traditional industry companies. Since the technical companies are more sensitive to $\mathbf{h_2}$, we suggest that $\mathbf{h_2}$ corresponds to the "Internet bubble" which largely affected technical companies during year 2000-2001.

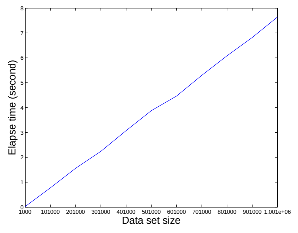
(Observation 2) *AutoSplit automatically discovered the meaningful underlying factors, namely, the general trend and the Internet bubble (Figure 6(b)).*

(Observation 3) *We found rules like "companies in financial and traditional industry grow steadily during 1990-2002, while technical companies suffered from the event around late-2000", and detected outliers like AT&T, which does not follow the general growth trend during the period 1990-2002 (Table 1(a)).*

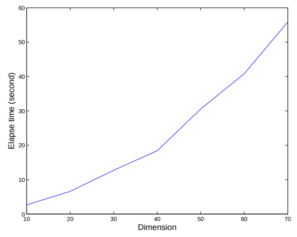
3.2 AutoSplit : Quality and Scalability



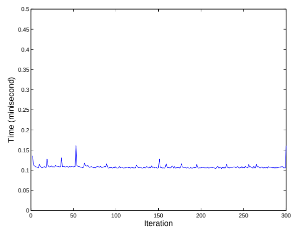
(a) "Broad jumps"



(b) Time vs data set size



(c) Time vs dimensionality



(d) Time per iteration

Fig. 7. (a): Bases found by Batch-AutoSplit (solid), PCA (dash), and AutoSplit (dash-dot) on the "broad jumps" data set. (b)(c)(d): Scalability of AutoSplit .

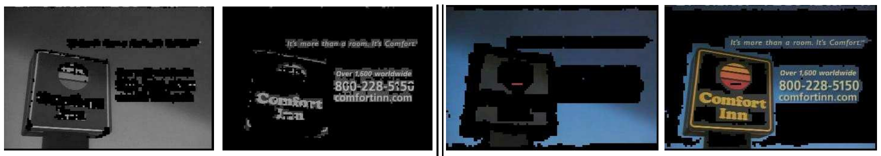
Can we operate AutoSplit on a continuous data stream? If yes, at what accuracy loss? Here we compare our "AutoSplit", the online processing algorithm, with the Batch-AutoSplit. Figure 7(a) shows the bases generated by AutoSplit, along with those by Batch-AutoSplit and PCA. Batch-AutoSplit has access to the complete data, so it gives near perfect bases (solid vectors). AutoSplit gives bases (dash-dot vectors) which are very close to the true bases.

We also studied the scalability of AutoSplit with respect to the data set size and to the data dimensionality. To study the effect of data set size, we generate 2-D synthetic data set similar to our "broad jumps" data set, but with more data points (vary from n=10^3 to 10^6). Figure 7(b) shows the total running time of AutoSplit (until convergence) versus the data set size. The total running time of AutoSplit is linear (O(n)) to the data set size (n), as expected, for the computational cost per iteration is constant (several matrix multiplications). Figure 7(d) shows the small constant computational cost per iteration ($\approx$ 0.12 msec), when AutoSplit is applied to the "broad jumps" data set.

To study the effect of dimensionality, we fixed the data set size to n=$35,000$, while varies the dimensionality from m=10 to 70. Synthetic data of higher dimensionality are generated to have a non-orthogonal distribution similar to a high-dimensional version of the "broad jumps" data set with more "spikes". The total running time (Figure 7(c)) of AutoSplit is super-linear, and probably quadratic (O(m^2)).

3.3 Experiment with Clustering-AutoSplit

We apply Clustering-AutoSplit to separate two texture classes in an image: overlay text and background. The idea is to find two different sets of bases for image patches of the two classes. Figure 8(a1)(a2) show Clustering-AutoSplit (with k=2) gives good separation of the overlay text from the background in a video frame [23]. The data items (each is a 36-dimensional row vector in $\mathbf{X}$) are the 6-by-6 pixel blocks taken from the frame. Figure 8(b1)(b2) show the failure of the PCA-based mixture model (MPPCA, Mixture of Probabilistic PCA [24]) on this task. MPPCA fails to differentiate the background edges with the real texts.



(a1) Background (a2) Overlay text (b1) Background (b2) Overlay text

Fig. 8. Texture segmentation Result from (a) Clustering-AutoSplit (b) MPPCA.

4 Conclusions

We propose AutoSplit, a powerful, incremental method for processing streams as well as static, multimedia data. The proposed "Clustering-AutoSplit" extends the feature discovery to multiple feature/bases sets and shows a better performance than the PCA-based method in texture segmentation (Figure 8). The strong points of AutoSplit are:

- It finds *bases* which better capture the natural trends and correlation of the data set (Figure 1(a)).
- It finds rules, which are revealed in the basis matrix $\mathbf{B}$ (Observation 1,3).
- It scales linearly with the number n of data points (Figure 7(b)).
- Its incremental, single-pass algorithm (Figure 3) makes it readily suitable for processing on streams.

References

1. Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., Harshman, R.A.: Indexing by latent semantic analysis. Journal of the American Society of Information Science **41** (1990) 391–497
2. Turk, M., Pentland, A.: Eigenfaces for recognition. Journal of Cognitive Neuroscience **3** (1991) 72–86

3. Duda, R.O., Hart, P.E., Stork, D.G.: Pattern Classification, 2nd Edition. New York: Wiley (2000)
4. Jolliffe, I.T.: Principal Component Analysis. Springer-Verlag (1986)
5. Korn, F., Labrinidis, A., Kotidis, Y., Faloutsos, C.: Ratio rules: A new paradigm for fast, quantifiable data mining. In: VLDB. (1998)
6. Garofalakis, M., Gehrke, J., Rastogi, R.: Querying and mining data streams: You only get one look. In: VLDB. (2002)
7. Guha, S., Gunopulos, D., Koudas, N.: Correlating synchronous and asynchronous data streams. In: SIGKDD 2003. (2003)
8. Kanth, K.V.R., Agrawal, D., Singh, A.K.: Dimensionality reduction for similarity searching in dynamic databases. In: SIGMOD. (1998) 166–176
9. Garofalakis, M., Gibbons, P.B.: Wavelet synopses with error guarantees. In: SIGMOD 2002. (2002)
10. Achlioptas, D.: Database-friendly random projections. In: PODS. (2001) 274–281
11. Indyk, P., Koudas, N., Muthukrishnan, S.: Identifying representative trends in massive time series data sets using sketches. In: Proc. VLDB. (2000) 363–372
12. Gunopulos, D., Das, G.: Time series similarity measures and time series indexing. In: SIGMOD. (2001) 624
13. Jensen, C.S., Snodgrass, R.T.: Semantics of time-varying information. Information Systems **19** (1994) 33–54
14. Teng, W.G., Chen, M.S., Yu, P.S.: A regression-based temporal pattern mining scheme for data streams. In: VLDB 2003. (2003) 93–104
15. Yi, B.K., Sidiropoulos, N.D., Johnson, T., Jagadish, H., Faloutsos, C., Biliris, A.: Online data mining for co-evolving time sequences. In: ICDE. (2000)
16. Jagadish, H., Mendelzon, A., Milo, T.: Similarity-based queries. In: PODS '95. (1995)
17. Moon, Y.S., Whang, K.Y., Han, W.S.: General match: a subsequence matching method in time-series databases based on generalized windows. In: SIGMOD 2002. (2002) 382–393
18. Keogh, E., Chakrabarti, K., Mehrotra, S., Pazzani, M.: Locally adaptive dimensionality reduction for indexing large time series databases. In: SIGMOD. (2001) 151–162
19. Korn, F., Jagadish, H.V., Faloutsos, C.: Efficiently supporting ad hoc queries in large datasets of time sequences. In: Proc. SIGMOD. (1997) 289–300
20. Lee, J., Chai, J., Reitsma, P.S.A., Hodgins, J.K., Pollard, N.S.: Interactive control of avatars animated with human motion data. In: SIGGRAPH 2002. (2002)
21. Hyvarinen, A., Karhunen, J., Oja, E.: Independent Component Analysis. John Wiley & Sons (2001)
22. Lewicki, M.S.: Estimating sub- and super-gaussian densities using ica and exponential power distributions with applications to natural images. Unpublished Manuscript (2000)
23. Wactlar, H., Christel, M., Gong, Y., Hauptmann, A.: Lessons learned from the creation and deployment of a terabyte digital video library. IEEE Computer **32** (1999) 66–73
24. Tipping, M., Bishop, C.: Mixture of probabilistic principal component analyzers. Neural Computation (1998)