Der SFA-Algorithmus für Klassifikation

Wolfgang Konen

Cologne University of Applied Sciences, CIOP Report 08/11

Last update: November 2011

Dieser Technische Report fasst den SFA-Algorithmus für Klassifikation zusammen, wie er im MATLAB-Paket **sfa-tk** ab Version V2.6 (aktuelle Version V2.8)[1] implementiert ist.
Für eine Einführung in SFA siehe [WisS02, Wis03c], für eine Einführung in sfa-tk siehe [Berkes03, Berkes05]. Für eine tiefgehendere Beschreibung der numerischen Verbesserungen an sfa-tk ab V2.6 (Stichwort SVD-SFA) siehe [Konen09b]. Für eine umfassendere Beschreibung des Parametric Bootstrap Algorithmus und seiner Verwendung in SFA siehe [Konen11].

SFA-TK: Algorithmus 'SVD_SFA'

Gegeben sei eine Menge von Datenvektoren $M_S=\{\mathbf{s}^{(1)},\ldots, \mathbf{s}^{(M)}\} \subset \mathbf{R}^m$.
Der Erwartungswert-Operator E[.] beschreibe die Mittelung über alle i=1,…,M.

Bei der Gestenklassifikation sind die Datenvektoren z.B. 90-dim. Vektoren, jeder ist ein Exemplar einer bestimmten Gestenklasse c=1,…,K.
Typische Werte sind `pp_range` =11, `xp_range`=11+ ½*11*12, K=6, `gaussdim` =K-1=5.

Variable	`SFA_STRUCTS{hdl}.`	Bedeutung
$\mathbf{s}^{(i)}$		Input-Vektor, Dim m
$\mathbf{s_0} = E[\mathbf{s}^{(i)}]$	`avg0`	Mittelwert
$\mathbf{W_0}$	`W0`	Whitening der Input-Daten, Zeilen von $\mathbf{W}_0$ sind Vielfache der EV zu Cov(**s**). $\mathbf{W}_0$ hat `pp_range` Zeilen.
$\mathbf{W_0}^{-1}$	`DW0`	Dewhitening der Input-Daten
	`D0`	Vektor der EW von Cov(**s**)
$\mathbf{D_0}$	`diag(D0)`	Diagonalmatrix der EW von Cov(**s**)
$\mathbf{x}^{(i)} = \mathbf{W_0}(\mathbf{s}^{(i)} - \mathbf{s_0})$		reduzierte Dim `pp_range`
$\mathbf{v}^{(i)} = \mathbf{h}(\mathbf{x}^{(i)})$		expandierter Vektor, Dim p=`xp_range`
$\mathbf{v_0} = E[\mathbf{v}^{(i)}]$	`avg1`	Mittelwert
$\mathbf{B} = \mathrm{Cov}(\mathbf{v})$	`xp_hdl.COV_MTX`	Kovarianzmatrix der expandierten Daten
$\mathbf{S}$	`myS`	Sphering der expandierten Daten, p x p, Zeilen von **S** sind Vielfache der EV zu $\mathbf{B} = \mathrm{Cov}(\mathbf{v})$. Es gilt $\mathbf{SBS}^T=\mathbf{1}$.
$\mathbf{D_B}$		Vektor `BD` der EW von Cov(**v**)
$\mathbf{z}^{(i)} = \mathbf{S}\,(\mathbf{v}^{(i)} - \mathbf{v_0})$		
$\mathbf{C}' = \mathrm{Cov}(\dot{\mathbf{v}})$	`diff_hdl.COV_MTX`	Kovarianzmatrix der „Ableitung" der expandierten Daten
$\mathbf{C} = \mathrm{Cov}(\dot{\mathbf{z}})$	$\mathbf{C} = \mathbf{SC'S}^T$	Kovarianzmatrix „Ableitung gesphered"
$\mathbf{W_1}$		die Spalten von $\mathbf{W}_1$ = `W1` sind Eigenvek-

[1] Download der MATLAB-Version: http://gociop.de/downloads/.

		toren zu $\mathrm{Cov}(\dot{\mathbf{z}})$
$\mathbf{D_1} = \mathbf{D}$	`D1, D`	Diagonalmatrix `D1` der EW von $\mathbf{C} = \mathrm{Cov}(\dot{\mathbf{z}})$
$[d_1\, d_2 \ldots d_p]$	`DSF`	Vektor der EW von $\mathrm{Cov}(\dot{\mathbf{z}})$, Dim p
$\mathbf{w}_j = (\mathbf{S}^T \mathbf{W}_1)_j$	`SF(j,:)`	j=1,…,G = `gaussdim`

Anmerkungen:

- Wenn `pp_range` <m, dann wird $\mathbf{W}_0$ auf die wichtigsten `pp_range` Zeilen reduziert, das sind diejenigen EV zu Cov(**s**) mit den größten Eigenwerten.
- $\mathbf{W}_0$ hat Null-Zeilen, wenn Cov(**s**) nicht vollen Rang hat, also Eigenwerte =0 hat. Diese Zeilen werden entfernt und `pp_range = min(pp_range,` # Nicht-Null-Zeilen`)`.
- Wenn **B** singulär oder schlecht konditioniert, sind einige Zeilen von **S** identisch Null.
- Die Eigenwerte in **D** und `DSF` werden der Größe nach aufsteigend sortiert, wobei Eigenwerte < `opts.epsC` $\cdot \lambda_{max}$ vorher entfernt werden. Allerdings ist für Klassifikation `opts.epsC` =0 die empfohlene Einstellung.
- Nach gleicher Systematik werden die $\mathbf{w}_j$ sortiert, so dass die ersten G Vektoren $\mathbf{w}_j$ die Richtungen der „gültigen“ langsamen SFA-Signale im expandierten Raum enthalten.

Der Begriff „Ableitung“ hat je nach Anwendungsart der SFA verschiedene Bedeutung:

- **method=“TIMESERIES“**: Dann sind die $\mathbf{s}^{(i)}$ Datenvektoren zu aufeinanderfolgenden Zeitschritten t_i. Entsprechend sind die $\mathbf{v}^{(i)}$ transformierte Datenvektoren zu den gleichen Zeitschritten t_i. Mit $\dot{\mathbf{v}}^{(i)}$, i=2,…,M als Ableitung bezeichnen wir die Menge der Differenzvektoren, also

  $$\dot{\mathbf{v}}^{(2)} = \mathbf{v}^{(2)} - \mathbf{v}^{(1)}, \quad \ldots \quad, \dot{\mathbf{v}}^{(M)} = \mathbf{v}^{(M)} - \mathbf{v}^{(M-1)}$$

  Die Kovarianzmatrix $\mathrm{Cov}(\dot{\mathbf{v}})$ wird aus allen $\dot{\mathbf{v}}^{(i)}$, i=2,…,M gebildet.
- **method=“CLASSIF“**: Dann sind die $\mathbf{s}^{(i)}$ Datenvektoren, die zu bestimmten Klassen c=1,…,K gehören. Für jede Klasse c=1,…,K bilden wir innerhalb einer Klasse mit allen möglichen „Pärchen“ jeweils 2-elementige Mini-Zeitreihen und berechnen für jede solche Mini-Zeitreihe einen Differenzvektor $\dot{\mathbf{v}}$: Sei

  $M_c = \{i \in \{1,\ldots,M\} \mid \mathbf{s}^{(i)}$ gehört zur Klasse c$\}$

  $V_c = \{\dot{\mathbf{v}} = \mathbf{v}^{(k)} - \mathbf{v}^{(k')} \mid k,k' \in M_c,\ k<k'\}$

  Die Kovarianzmatrix $\mathrm{Cov}(\dot{\mathbf{v}})$ wird aus allen $\dot{\mathbf{v}} \in V_1 \cup V_2 \cup \ldots \cup V_K$ gebildet.

SFA trainieren

Der verbesserte Algorithmus ‚SVD_SFA’, der **z** und $\dot{\mathbf{z}}$ nicht explizit berechnen muss, läuft im Training in **sfaClassModel.m** (innere Fkt. **sfa_step.m, sfa_execute.m**) wie folgt ab:

(a) Zu Input $\mathbf{s}^{(i)}$, i=1,…,M bestimme $\mathbf{W}_0$ und $\mathbf{s}_0$. und damit $\mathbf{x}^{(i)} = \mathbf{W}_0(\mathbf{s}^{(i)}-\mathbf{s}_0)$

(b) Expandiere $\mathbf{v}^{(i)} = \mathbf{h}(\mathbf{x}^{(i)})$ und bestimme $\mathbf{v}_0$

(c) Bilde **B, S** und parallel **C**’ = $\mathrm{Cov}(\dot{\mathbf{v}})$. Wenn **B** singulär oder schlecht konditioniert, dann sind einige Zeilen von **S** identisch Null.

(d) Bestimme für $\mathbf{C} = \mathbf{SC'S}^T = \mathrm{Cov}(\dot{\mathbf{z}})$ die Eigenvektoren (Spalten von $\mathbf{W}_1$).

(e) Setze $\mathbf{w}_j = (\mathbf{S}^T \mathbf{W}_1)_j$ (j. Spalte). Entferne die Spalten von $\mathbf{W}_1$, die identisch Null sind.[2] Sortiere die $\mathbf{w}_j$ nach aufsteigenden Eigenwerten von **C**.

(f) Speichere { $\mathbf{W}_0$, $\mathbf{s}_0$, $\mathbf{v}_0$, $\mathbf{w}_j$, j=1,…,G }. Die ersten G Vektoren $\mathbf{w}_j$ geben die Richtungen im expandierten Raum an, die langsamen SFA-Signalen entsprechen.

[2] sowie fallweise die, deren zugehörigen Eigenwerte in **C** im Rahmen der numerischen Genauigkeit Null sind (d.h. < `opts.epsC` $\cdot \lambda_{max}$). Siehe allerdings auch Anmerkung zu `opts.epcsC`.

SFA anwenden

Damit ist das Training beendet und die langsamen Signale y_j können wie folgt mit **sfaClassPredict.m** (innere Fkt. **sfa_execute.m**) berechnet werden, wobei **s** entweder ein Datenvektor aus den Trainingsdaten oder ein Datenvektor aus neuen (Test)-Daten ist:

(a) Lade { $\mathbf{W}_0$, $\mathbf{s}_0$, $\mathbf{v}_0$, $\mathbf{w}_j$, j=1,…,G }
(b) $\mathbf{x} = \mathbf{W}_0(\mathbf{s}-\mathbf{s}_0)$
(c) Expandiere $\mathbf{v} = \mathbf{h}(\mathbf{x})$
(d) $y_j = \mathbf{w}_j^T (\mathbf{v}-\mathbf{v}_0)$, j=1,…,G.

Der neue Vektor $\mathbf{y} = (y_1,\ldots,y_G)^T$ sollte gut geeignet sein, um die Klasse des Datenvektors **s** zu bestimmen. Zur Klassifikation kann entweder ein Gauss-Klassifikator oder ein Nearest-Neighbor-Klassifikator (für kleinere Mengen von Trainingsvektoren) benutzt werden.

Klassifikatoren

Gauss-Klassifikator trainieren

Gegeben sei eine Menge von Klassifikationsvektoren $M_Y=\{\mathbf{y}^{(1)},\ldots,\mathbf{y}^{(M)}\} \subset \mathbf{R}^G$. Von diesen gehören manche zur Klasse c=1, manche zu c=2, …., manche zu c=K. Sei

$M_c=\{i\in\{1,\ldots,M\} \mid \mathbf{s}^{(i)}$ gehört zur Klasse c$\}$

Die Häufigkeit, mit der Vektoren der Klasse c in der Trainingsmenge auftreten, sei ein Maß für die a-priori-Wahrscheinlichkeit dieser Klasse, d.h. $P(c) = |M_c|/|M_Y|$.

Variable	`GAUSS_STRUCTS{hdl}.`	Bedeutung
$\mathbf{y}^{(i)}$		Klass.-Vektor, Dim G = `gaussdim`
$\mathbf{y}_{0,c} = E[\mathbf{y}^{(i)} \mid i\in M_c]$	`X0(c,:)`	Mittelwert der Daten zu Klasse c
$\mathbf{a}_c^{(i)} = \mathbf{y}^{(i)} - \mathbf{y}_{0,c}$		mittelwertbereinigter Klass.-Vektor
$\mathbf{G}_c = \mathrm{Cov}(\mathbf{y}^{(i)} \mid i\in M_c)$	`COV(:,:,c)`	Kovarianzmatrix der Daten zu Klasse c
$\mathbf{G}_c^{-1}$	`iCOV(:,:,c)`	Inverse Kovarianzmatrix
P(c)	`P_c(c)`	a-priori-Wahrsch. $P(c)=\|M_c\|/M$ für Klasse c
$f_c = (2\pi)^{-G/2}(\det \mathbf{G}_c)^{-1/2}$	`f0(c)`	Vorfaktor für Klasse c

Der Gauss-Klassifikator wird in **gaussClassifier.m** trainiert (`method='train', algo='gauss'`) :

(a) Bilde { $\mathbf{y}_{0,c}$, $\mathbf{G}_c$, $\mathbf{G}_c^{-1}$, P(c), f_c | c=1,…,K }.
(b) Wenn `aligned=1`: Setze in $\mathbf{G}_c$ alle Off-Diagonalelemente auf 0 und berechne $\mathbf{G}_c^{-1}$ erneut. Diese Version ist numerisch stabiler, kann sich aber nicht so gut an „schiefliegende“ Datenverteilungen anpassen.
(c) Speichere { $\mathbf{y}_{0,c}$, $\mathbf{G}_c$, $\mathbf{G}_c^{-1}$, P(c), f_c | c=1,…,K }.

Gauss-Klassifikator anwenden

Der Gauss-Klassifikator wird in **gaussClassifier.m** auf einen (neuen) Datenvektor **y** angewendet (`method='apply'`):

(a) Lade { $\mathbf{y}_{0,c}$, $\mathbf{G}_c$, $\mathbf{G}_c^{-1}$, P(c), f_c | c=1,…,K }
(b) Bilde für jedes c=1,…,K:
(1) $\mathbf{a}_c = \mathbf{y} - \mathbf{y}_{0,c}$
(2) $P(\mathbf{y} \mid c) = f_c \exp\left(-\frac{\mathbf{a}_c^T \mathbf{G}_c^{-1} \mathbf{a}_c}{2}\right)$

$$(3)\quad P(c \mid \mathbf{y}) = \frac{P(\mathbf{y} \mid c)P(c)}{\sum_{c'=1}^{K} P(\mathbf{y} \mid c')P(c')}$$

(c) Liefere die Klasse $c^*=\arg\max_c P(c|\mathbf{y})$ zurück.

Nearest-Neighbor-Klassifikator trainieren

Der Nearest-Neighbor-Klassifikator wird in **gaussClassifier.m** trainiert (`method='train', algo='nearneig'`) :

Gegeben sei eine Menge von Klassifikationsvektoren $M_Y=\{\mathbf{y}^{(1)},\ldots, \mathbf{y}^{(M)}\} \subset \mathbf{R}^G$. Von diesen gehören manche zur Klasse c=1, manche zu c=2, …., manche zu c=K.

Das „Training" besteht in diesem Fall einfach aus der Speicherung der Klassifikationsvektoren und ihrer Klassenzugehörigkeiten, also einer Menge von Paaren

$$\{ (\mathbf{y}^{(i)},c^{(i)}) \mid i=1,\ldots,M \}$$

Nearest-Neighbor-Klassifikator anwenden

Ein (neuer) Datenvektor **y** wird wie folgt klassifiziert:
Bestimme den naheliegendsten gespeicherten Vektor $\mathbf{y}^{(i)}$ und liefere dessen Klasse zurück:

$$c^* = c^{(i^*)} \quad \text{mit} \quad i^* = \arg\min_i \|\mathbf{y} - \mathbf{y}^{(i)}\|^2$$

Literatur

- [WisS02] Wiskott, L. and Sejnowski, T.J. (2002), **Slow Feature Analysis: Unsupervised Learning of Invariances**, Neural Computation, 14(4):715-770
- [Wis03c] Wiskott, L. **Estimating driving forces of nonstationary time series with slow feature analysis.** arXiv.org e-Print archive, http://arxiv.org/abs/cond-mat/0312317, December 2003.
- [Berkes05] Berkes, P. **Pattern recognition with Slow Feature Analysis.** Cognitive Sciences EPrint Archive (CogPrint) 4104, http://cogprints.org/4104/ (2005).
- [Berkes03] Berkes, P. **SFA-TK: Slow Feature Analysis Toolkit** for Matlab (v.1.0.1). http://itb.biologie.hu-berlin.de/~berkes/software/sfa-tk/sfa-tk.shtml or http://people.brandeis.edu/~berkes/software/sfa-tk/index.html.
- [Konen09a] Konen, W., Koch, P. (2009). **How slow is slow? SFA detects signals that are slower than the driving force**. arXiv.org e-Print archive, http://arxiv.org/abs/0911.4397v1, November 2009 (PDF)
- [Konen09b] Konen, W. (2009). **On the numeric stability of the SFA implementation `sfa-tk`**. arXiv.org e-Print archive, http://arxiv.org/abs/0912.1064, December 2009. (PDF)
- [Koch10a] Koch, P., Konen, W., Hein, K., Gesture Recognition on Few Training Data using Slow Feature Analysis and Parametric Bootstrap. In P. Sobrevilla (ed.), Proc. IEEE World Congress on Computational Intelligence (WCCI), Barcelona, July 2010. (PDF)
- [Konen11] Konen, W. (2011). SFA classification with few training data: Improvements with parametric bootstrap. CIOP Technical Report 09/2011, Cologne University of Applied Sciences.

ANHANG A: Kovarianzmatrix, Sphering-Matrix

Gegeben sei eine Menge von Datenvektoren $V=\{\mathbf{v}^{(1)},\ldots,\mathbf{v}^{(M)}\} \subset \mathbf{R}^m$.
Der Erwartungswert-Operator E[.] beschreibe die Mittelung über alle i=1,…,M.
Der Mittelwert der Datenvektoren sei $\mathbf{v}_0 = E[\mathbf{v}^{(i)}]$.
Die **Kovarianzmatrix** ist definiert durch

$$\mathbf{B} = \mathrm{Cov}(\mathbf{v}) = E[(\mathbf{v}^{(i)} - \mathbf{v}_0)(\mathbf{v}^{(i)} - \mathbf{v}_0)^T] = E[\mathbf{v}^{(i)}\mathbf{v}^{(i)^T}] - \mathbf{v}_0\mathbf{v}_0^T$$

Die letzte Identität ergibt sich, wenn man den Mittelwert auf die einzelnen Terme durchzieht. Man beachte, dass bei der Definition der Kovarianz die Subtraktion des Mittelwertes „$-\mathbf{v}_0$“ nicht fehlen darf.

I.d.R. wird die Kovarianzmatrix nicht die Einheitsmatrix sein, sondern

(a) Off-Diagonalelemente haben, was anzeigt, dass es Korrelationen zwischen den Datendimensionen (Variablen) v_i und v_k gibt,
(b) die Diagonalelemente werden nicht alle gleich sein, was anzeigt, dass in einigen Variablen mehr Varianz steckt als in anderen.

Sphering, Whitening: Man möchte nun manchmal „gespherte“ oder „weiße“ Daten $\mathbf{z}^{(i)}$, die aus den Originaldaten durch eine lineare Transformation hervorgehen:[3]

$$\mathbf{z}^{(i)} = \mathbf{S}(\mathbf{v}^{(i)} - \mathbf{v}_0) \quad \text{mit} \quad E[\mathbf{z}^{(i)}] = \mathbf{z}_0 = 0 \quad \text{und} \quad \mathrm{Cov}(\mathbf{z}) = E[\mathbf{z}^{(i)}\mathbf{z}^{(i)^T}] = \mathbf{1}$$

Hierbei bezeichnet **1** die (m x m)-Einheitsmatrix. Die Mittelwertfreiheit der $\mathbf{z}^{(i)}$ ergibt sich unabhängig von **S** sofort aus dem Term ($\mathbf{v}^{(i)}$-$\mathbf{v}_0$). In Cov(**z**) haben wir die Terme „$-\mathbf{z}_0$“ gleich weggelassen, weil ja ohnehin $\mathbf{z}_0 = 0$ gilt.
Wie muss **S** aussehen?
Behauptung: Wenn **R** die Eigenvektormatrix von **B**=Cov(**v**) ist, dann gilt

$$\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{R} = \begin{pmatrix} 1/\sqrt{\lambda_1} & & \\ & \ddots & \\ & & 1/\sqrt{\lambda_m} \end{pmatrix} \begin{pmatrix} \cdots & \mathbf{r}_1^T & \cdots \\ & \cdots & \\ \cdots & \mathbf{r}_m^T & \cdots \end{pmatrix} = \begin{pmatrix} \cdots & \mathbf{r}_1^T/\sqrt{\lambda_1} & \cdots \\ & \cdots & \\ \cdots & \mathbf{r}_m^T/\sqrt{\lambda_m} & \cdots \end{pmatrix}$$

Hierbei ist $\mathbf{r}_k^T$, die k-te Zeile von **R**, der als Zeilenvektor geschriebene Eigenvektor von **B** zum Eigenwert λ_k, d.h. es gilt

$$\mathbf{B}\mathbf{r}_k = \lambda_k \mathbf{r}_k \quad \text{für} \quad k = 1,\ldots,m$$

D ist die Diagonalmatrix der Eigenwerte und $\mathbf{D}^{-1/2}$ steht für die Diagonalmatrix mit $\lambda_k^{-1/2}$ im k-ten Diagonalelement. In MATLAB erhält man **D** und **R** aus der SVD-Zerlegung, s. zum Beispiel in **lcov_pca2.m**

```
[tmp,D,R]=svd(LCOV_STRUCTS{handle}.COV_MTX);
```

ACHTUNG: Wenn es Eigenwerte λ_k=0 gibt, dann muss man mit $\mathbf{D}^{-1/2}$ aufpassen. In diesem Fall ist die übliche SVD-Behandlung in **lcov_pca2.m**, dass in $\mathbf{D}^{-1/2}$ jedes 1/0 durch 0 ersetzt wird. Dadurch werden dann einige Zeilen von **S** zu Nullzeilen.

Beweis der Behauptung $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{R}$ (im Falle, dass es keine Eigenwerte λ_k=0 gibt):

---

[3] Zur Namensgebung: Durch die lineare Transformation wird eine ellipsoidförmige Punktwolke in eine kreisförmige Punktwolke überführt (→ „sphering“). Eine ursprünglich in vielleicht einer Dimension „flache“ Datenverteilung wird dann in jeder Dimension die gleiche Varianz haben. Die transformierte Punktwolke wird ähnlicher zu „white noise“, d.h. einer Punktwolke mit Deltafunktion als Autokorrelation. Für spektrale Signale ist „white noise“ ein zufälliges Signal, das überall gleiche Spektraldichte hat, daher der Name „whitening“.

$$
\begin{aligned}
E[\mathbf{z}\mathbf{z}^T] &= \mathbf{D}^{-1/2}\mathbf{R}\cdot E[(\mathbf{v}-\mathbf{v}_0)(\mathbf{v}-\mathbf{v}_0)^T]\cdot\mathbf{R}^T\mathbf{D}^{-1/2} \\
&= \mathbf{D}^{-1/2}\underbrace{\mathbf{R}\cdot \mathrm{Cov}(\mathbf{v})\cdot\mathbf{R}^T}_{=\mathbf{D}}\mathbf{D}^{-1/2} \\
&= \mathbf{1}
\end{aligned}
$$

Die Umformung in der geschweiften Klammer gilt deshalb, weil

$$
\mathrm{Cov}(\mathbf{v})\cdot\mathbf{R}^T = \mathrm{Cov}(\mathbf{v})\begin{pmatrix} \vdots & & \vdots \\ \mathbf{r}_1 & \cdots & \mathbf{r}_m \\ \vdots & & \vdots \end{pmatrix} = \begin{pmatrix} \vdots & & \vdots \\ \mathbf{r}_1\lambda_1 & \cdots & \mathbf{r}_m\lambda_m \\ \vdots & & \vdots \end{pmatrix} = \mathbf{R}^T\mathbf{D}
$$

Multipliziert man dies von links mit **R** durch, so erhält man

$$\mathbf{R}\cdot\mathrm{Cov}(\mathbf{v})\cdot\mathbf{R}^T = \mathbf{D}$$

wie behauptet.
Für die Sphering-Matrix **S** gilt ferner die Relation

$$\mathbf{S}\cdot\mathrm{Cov}(\mathbf{v})\cdot\mathbf{S}^T = \mathbf{1}$$

denn

$$
\begin{aligned}
\mathbf{S}\cdot\mathrm{Cov}(\mathbf{v})\cdot\mathbf{S}^T &= \begin{pmatrix} \cdots & \mathbf{r}_1^T/\sqrt{\lambda_1} & \cdots \\ & \cdots & \\ \cdots & \mathbf{r}_m^T/\sqrt{\lambda_m} & \cdots \end{pmatrix}\cdot\mathrm{Cov}(\mathbf{v})\cdot\begin{pmatrix} \vdots & & \vdots \\ \mathbf{r}_1/\sqrt{\lambda_1} & \cdots & \mathbf{r}_m/\sqrt{\lambda_m} \\ \vdots & & \vdots \end{pmatrix} \\
&= \begin{pmatrix} \cdots & \mathbf{r}_1^T/\sqrt{\lambda_1} & \cdots \\ & \cdots & \\ \cdots & \mathbf{r}_m^T/\sqrt{\lambda_m} & \cdots \end{pmatrix}\cdot\begin{pmatrix} \vdots & & \vdots \\ \lambda_1\mathbf{r}_1/\sqrt{\lambda_1} & \cdots & \lambda_m\mathbf{r}_m/\sqrt{\lambda_m} \\ \vdots & & \vdots \end{pmatrix} = \mathbf{1}
\end{aligned}
$$

aufgrund der Orthonormalität $\mathbf{r}_i^T\mathbf{r}_k = \delta_{ik}$.

ANHANG B: Ablaufplan MATLAB-Files sfa-tk V2.7 für Klassifikation

- class_demo2A_2B.m
  - class_demo2A.m
    - dataLoad.m (test + training)
    - sfaClassModel.m
      - sfa2_create.m
      - sfa_step.m → sfa_step2.m
      - sfa_save.m
      - sfa_execute.m
      - gaussCreate.m
      - gaussClassifier.m (‘train‘)
      - gaussSave.m
    - mk_confmat.m
    - save test & training data
  - class_demo2B.m
    - load test & training data
    - sfaClassPredict.m
      - sfa_load.m
      - gaussLoad.m
      - sfa_execute.m
      - gaussClassifier.m (‘apply‘)
    - mk_confmat.m