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, `gaussdim` =K-1.

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
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 Eigenvektoren zu $\mathrm{Cov}(\dot{\mathbf{z}})$
$\mathbf{D_1} = \mathbf{D}$	`D1, D`	Diagonalmatrix `D1` der EW von $\mathrm{Cov}(\dot{\mathbf{z}})$
$[d_1\, d_2 \ldots d_p]$	`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`

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 mit allen möglichen „Pärchen" innerhalb einer Klasse 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 $\mathbf{z}$ 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 $\mathbf{C}' = \mathrm{Cov}(\dot{\mathbf{v}})$. Wenn **B** singulär, 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.
(f) Speichere { $\mathbf{W}_0$, $\mathbf{s}_0$, $\mathbf{v}_0$, $\mathbf{w}_j$, j=1,…,G }

SFA anwenden

Damit ist das (unüberwachte) 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.

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'`) :

(a) Bilde $\mathbf{y}_{0,c}$, $\mathbf{G}_c$, $\mathbf{G}_c^{-1}$, P_c, f_c für 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) $\mathbf{a}_c^{(i)} = \mathbf{y}^{(i)} - \mathbf{y}_{0,c}$

(2) $$P(\mathbf{y} \mid c) = f_c \exp\left(-\frac{\mathbf{a}_c^{(i)T} \mathbf{G}_c^{(i)} \mathbf{a}_c^{(i)}}{2} \right)$$

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

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

Nearest-Neighbor-Klassifikator trainieren

(Dieser Klassifikator ist nicht in MATLAB implementiert. Er stellt einen einfachen Klassifikator für kleine Trainingsmengen dar.)

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:

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

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: Man möchte nun manchmal „gespherte“ Daten $\mathbf{z}^{(i)}$ (zur Namensgebung s. hier in notes_data.doc) haben, die aus den Originaldaten durch eine lineare Transformation hervorgehen:

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

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** ist, dann gilt

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

Hierbei ist $\mathbf{r}_k^{\mathrm{T}}$, 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 $\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{R}$ ist hier in notes_SFA.doc nachzulesen.

ANHANG B: Ablaufplan MATLAB-Files 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