DIMENSIONSREDUKTION

Joern Ploennigs · AI4SC

cf. Edvard Munch "The Scream"

Dimensionsreduktion | Motivation und Einsatzfelder

Methoden zur *Dimensionsreduktion* werden im Machine Learning immer dann genutzt, wenn Datensätze vereinfacht werden sollen, um sie besser zu verstehen oder schneller zu erlernen. Hierbei teilt man die Verfahren in zwei Anwendungsfälle:

- Reduktion der Anzahl an Variablen in sehr großen Datensätzen mittels der **Hauptkomponentenanalyse**
- Reduktion der Anzahl an Beobachtungen, um typische Muster abzuleiten mittels **Matrixfaktorisierung**

HAUPTKOMPONENTENANALYSE (PCA)

Methode

Die Hauptkomponentenanalyse (en. *Principal Component Analysis*, PCA) wird eingesetzt, um Datensätze mit vielen Variablen zu vereinfachen, damit wir sie besser interpretieren und nutzen können und damit sich kompaktere Modelle trainieren lassen. Das ist insbesondere bei Datensätzen mit vielen, stark korrelierten Variablen sinnvoll, da sowohl Nutzer die vielen Variablen nur schwer verstehen als auch viele lineare Modelle bei stark korrelierten Variablen instabil werden oder zusätzliche Regularisierung benötigen.

PCA | Korrelierte Sensoren verdichten

Stellen wir uns vor, wir überwachen eine Brücke mit 10 verschiedenen Sensoren. Viele der Messreihen beschreiben ähnliche Effekte und sind stark korreliert. PCA hilft dabei, die wichtigsten Muster, z.B. eine Vibration in X/Y-Zugrichtung, aus diesen Daten herauszufiltern und auf 2 bis 3 Hauptachsen zu reduzieren.

Wir betrachten den bekannten Energie- und Wetter Datensatz.

```
import numpy as np # Import von NumPy
import pandas as pd # Import von Pandas

egywth = pd.read_csv("../data/UROS/Energy1D_weather_clean.csv", parse_dates=[0])
egywth.head()
```

	Date	EV_HT_740	EV_NT_740	E_AV_Lab	E_SV_Lab	ES_Lab	DATUM_DT	STATIONS_ID	MESS_DATUM	QN_3	...	TMK	UPM	TXK	TNK	TGK	eor	TemperaturKlasse	HeizKuehlTage
0	2020-12-30 00:00:00+00:00	NaN	NaN	NaN	NaN	NaN	2020-12-30 00:00:00+00:00	4271.0	20201230.0	10.0	...	4.0	81.0	4.6	2.9	2.2	eor	Cold	Heizgradtag
1	2020-12-31 00:00:00+00:00	NaN	NaN	1256.0	291.0	5.0	2020-12-31 00:00:00+00:00	4271.0	20201231.0	10.0	...	3.4	83.0	4.4	2.3	0.9	eor	Cold	Heizgradtag
2	2021-01-01 00:00:00+00:00	0.0	4080.0	1221.0	290.0	1.0	2021-01-01 00:00:00+00:00	4271.0	20210101.0	10.0	...	2.0	90.0	3.0	1.1	0.3	eor	Cold	Heizgradtag
3	2021-01-02 00:00:00+00:00	1170.0	2630.0	1243.0	284.0	2.0	2021-01-02 00:00:00+00:00	4271.0	20210102.0	10.0	...	3.3	95.0	4.0	2.6	2.0	eor	Cold	Heizgradtag
4	2021-01-03 00:00:00+00:00	0.0	3750.0	1222.0	283.0	2.0	2021-01-03 00:00:00+00:00	4271.0	20210103.0	10.0	...	3.5	81.0	4.6	2.4	0.8	eor	Cold	Heizgradtag

5 rows × 30 columns

PCA-Beispiel | Numerische Variablen auswählen

Wir analysieren vom Datensatz allerdings nur die numerischen, nicht-`NA` Spalten, weshalb wir sie selektieren.

```
egywthNumber = egywth.select_dtypes(include='number').dropna()
egywthNumber = egywthNumber[['EV_HT_740', 'EV_NT_740', 'E_AV_Lab', 'E_SV_Lab', 'ES_Lab', 'FX', 'FM', 'RSK', 'RSKF', 'SDK', 'SHK_TAG', 'NM', 'VPM', 'PM', 'TMK', 'UPM', 'TXK', 'TNK', 'TGK']]
```

PCA Schritt 1 | Daten um den Mittelwert zentrieren

Schritt 1 - Datenzentrierung: Die Daten werden zentriert, indem der Mittelwert jeder Dimension subtrahiert wird. Dadurch liegen alle Daten um den Ursprung herum.

$$X_{\text{cent}} = X - \mu_X$$

```
egywthCentered = egywthNumber - egywthNumber.mean()
egywthCentered.head()
```

	EV_HT_740	EV_NT_740	E_AV_Lab	E_SV_Lab	ES_Lab	FX	FM	RSK	RSKF	SDK	SHK_TAG	NM	VPM	PM	TMK	UPM
2	-1245.840611	1258.984716	27.776201	16.925764	-37.293668	-5.778166	-2.222162	1.699017	3.745633	-4.973059	-0.177948	2.145524	-3.410153	-8.547838	-7.610699	12.
3	-75.840611	-191.015284	49.776201	10.925764	-36.293668	-5.578166	-3.022162	-0.800983	3.745633	-4.973059	-0.177948	2.145524	-2.510153	-0.947838	-6.310699	17.
4	-1245.840611	928.984716	28.776201	9.925764	-36.293668	0.521834	-0.022162	-1.400983	1.745633	-4.973059	-0.177948	2.045524	-3.410153	4.852162	-6.110699	3.6
5	1994.159389	-1711.015284	69.776201	21.925764	-33.293668	2.521834	2.177838	-1.100983	1.745633	-4.973059	-0.177948	2.045524	-3.110153	5.752162	-6.010699	6.6
6	2224.159389	-1701.015284	152.776201	25.925764	-37.293668	3.721834	2.377838	1.499017	3.745633	-4.973059	-0.177948	2.145524	-2.810153	5.252162	-5.910699	10.

PCA Schritt 2 | Kovarianz und Korrelation erfassen

Schritt 2 - Berechnung der Kovarianzmatrix: Die Kovarianzmatrix C wird berechnet, um die Streuung und Korrelation zwischen den verschiedenen Dimensionen zu erfassen.

$$\boldsymbol{C} = \frac{\boldsymbol{X}_{\text{cent}}^T \boldsymbol{X}_{\text{cent}}}{n-1}.$$

```
cov_matrix = np.cov(egywthCentered, rowvar=False)
```

PCA Schritt 3 | Hauptrichtungen aus Eigenvektoren ableiten

Schritt 3 - Eigenwertzerlegung der Kovarianzmatrix: Die Kovarianzmatrix wird einer Eigenwertzerlegung unterzogen, um die Eigenwerte und Eigenvektoren zu berechnen. Die Eigenvektoren repräsentieren die Richtungen der größten Varianz, und die Eigenwerte repräsentieren die Größe der Varianz in diesen Richtungen. Der Eigenvektor einer quadratischen Matrix ist ein Vektor, der durch die Matrix nur um einen skalaren Faktor, den Eigenwert λ, gestreckt oder gestaucht wird. Mathematisch ausgedrückt:

$$\boldsymbol{C}\boldsymbol{v} = \lambda\boldsymbol{v}$$

dabei ist λ der skalare Eigenwert und $\boldsymbol{v}$ der zugehörige Eigenvektor.

```
# Berechne die Eigenwerte und Eigenvektoren
eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)
```

PCA Schritt 4 | Die wichtigsten Komponenten auswählen

Schritt 4 - Auswahl der Hauptkomponenten: Die Hauptkomponenten (*Principal Components*) sind die Eigenvektoren, die den größten Eigenwerten entsprechen. Diese Eigenvektoren definieren die neuen Achsen des transformierten Datenraums. Hierfür wählen wir die besten n Komponenten in der Matrix $\boldsymbol{W}$.

```
# Sortiere die Eigenvektoren nach absteigenden Eigenwerten
idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]

# Wähle die n Hauptkomponenten
n_components = 2
eigenvalues_sub = eigenvalues[:n_components]
W = eigenvectors[:, :n_components]
```

PCA Schritt 5 | Daten in den neuen Raum projizieren

Schritt 5 - Transformation der Daten: Die ursprünglichen Daten werden auf die neuen Hauptkomponenten projiziert, um die transformierten Daten zu erhalten.

$$\boldsymbol{X}_{\text{pca}} = \boldsymbol{X}_{\text{cent}} \cdot \boldsymbol{W}$$

```
# Transformiere die Daten
egywth_pca = np.dot(egywthCentered, W)
egywth_pca
```

```
array([[1764.35300097, -144.63239642],
       [ -70.67495979,  177.86671771],
       [1547.04272956,   99.44054371],
       ...,
       [ 920.66526878,  888.17908265],
       [ 914.15695   ,  895.99134945],
       [ 803.76433239, 1030.65117329]], shape=(916, 2))
```

PCA mit SciKit-Learn

Wichtig in der Praxis ist, dass PCA empfindlich gegenüber der Skalierung der Variablen ist. Variablen mit deutlich unterschiedlichen Skalen oder Größenordnungen können die ersten Hauptkomponenten sonst dominieren. Deshalb standardisiert man die Daten häufig vor einer PCA, wenn die Variablen unterschiedliche Einheiten oder Größenordnungen haben.

PCA in sklearn | Fit und Transformation ausführen

Für die Praxis gibt es dafür auch eine einfache Implementierung in `sklearn`, die wir wie gewohnt einsetzen. Wir können zum Training `fit` benutzen. Wollen wir allerdings gleich die Komponenten erhalten, bietet sich alternativ `fit_transform` an:

```
from sklearn.decomposition import PCA

pcam = PCA(n_components=2)
components = pcam.fit_transform(egywthNumber)
```

PCA in sklearn | Singulärwerte als Stärke der Komponenten

Wir können uns mit dem Attribut `explained_variance_` die Varianz der einzelnen Hauptkomponenten ausgeben lassen. Diese Werte entsprechen den Eigenwerten der Kovarianzmatrix. Das Attribut `singular_values_` enthält dagegen die Singulärwerte der zentrierten Datenmatrix aus der SVD, die `sklearn` intern zur Berechnung der PCA verwendet.

```
pcam.singular_values_
```

```
array([61197.4192215 , 11416.98508508])
```

PCA in sklearn | Erklärte Varianz vergleichen

Wir können uns auch mit `explained_variance_ratio_` die erklärte Varianz ausgeben lassen.

```
print(pcam.explained_variance_ratio_)
```

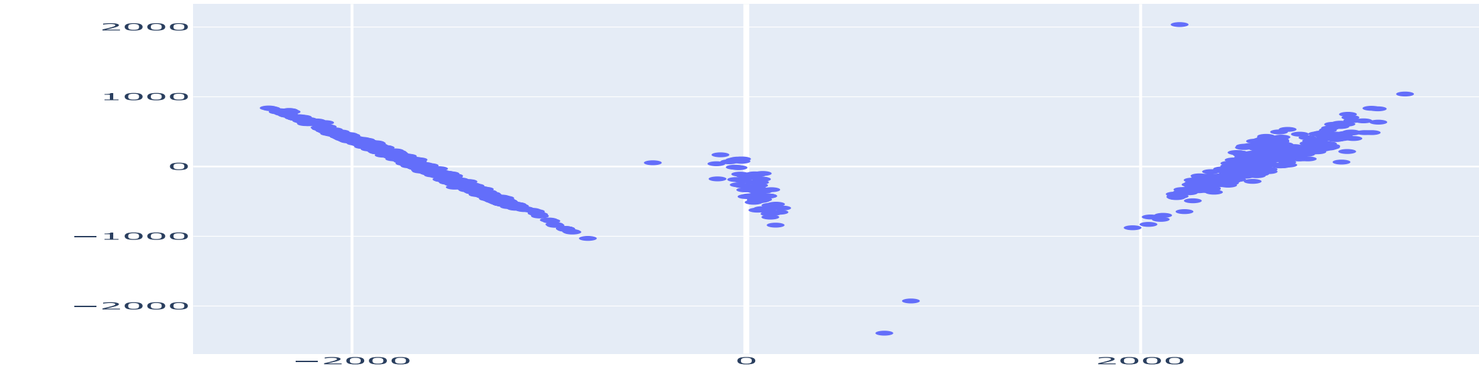
```
[0.96270602 0.0335066 ]
```

Hier sehen wir, dass die erste Komponente mit 96.7% den Großteil der Varianz erklärt, die zweite mit 3% deutlich weniger.

PCA in sklearn | Komponenten im Streudiagramm lesen

Als nächstes wollen wir die Komponenten visualisieren. Das ist ein weiterer wichtiger Anwendungsbereich von PCA. Da sich Datensätze mit sehr vielen Variablen schlecht visualisieren lassen, kann man sie mit PCA auf eine 2D- oder 3D-Darstellung reduzieren. Dies wird z.B. häufig beim Clustering verwendet, um höherdimensionale Cluster zu visualisieren.

```
import plotly.express as px
# Visualisierung der transformierten Daten
fig = px.scatter(components, x=0, y=1, width=500, height=500)
fig.update_layout( xaxis_title="Hauptkomponente 1", yaxis_title="Hauptkomponente 2")
# Visualisierung der Eigenvektoren
loadings = pcam.components_.T * np.sqrt(pcam.explained_variance_)
for i, feature in enumerate(egywthCentered.columns):
    fig.add_annotation(ax=0, ay=0, axref="x", ayref="y", x=loadings[i, 0], y=loadings[i, 1], showarrow=True, arrowsize=2, arrowhead=2, xanchor="right", yanchor="top")
    fig.add_annotation( x=loadings[i, 0], y=loadings[i, 1], ax=0, ay=0, xanchor="center", yanchor="bottom", text=feature, yshift=5 )
fig.show()
```



Welche Merkmale laden besonders stark auf die erste Hauptkomponente, und was sagt das über deren physikalische Bedeutung aus? Im Diagramm erkennen wir drei Gruppen um `"EV_NT_740"`, `"EV_HT_740"` und `"E_AV_Lab"`. Um letztere gruppieren sich auch die gesamten Wetterwerte. Das deutet darauf hin, dass diese deutlich stärker mit `"E_AV_Lab"` korreliert sind als mit den anderen Werten.

NICHTNEGATIVE MATRIXFAKTORISIERUNG (NMF)

Methode

Matrixfaktorisierung wird genutzt, um die Anzahl der Beobachtungen (die Anzahl der Variablen bleibt gleich) zu reduzieren. Dabei zerteilt man einen Datensatz in zwei niedrigdimensionale Matrizen. Das ist sinnvoll, wenn man die Daten komprimieren oder latente Merkmale extrahieren möchte. Es wird z.B. bei der Bilderkennung benutzt, aber auch als Alternative zum Clustering, wenn keine klaren Cluster erkennbar sind.

Die *nichtnegative Matrixfaktorisierung* ist ein typischer Ansatz, bei dem alle Elemente der Matrizen nichtnegativ sind (sein müssen). Dies ist besonders nützlich für Anwendungen, bei denen die Daten nichtnegativ sind, wie z.B. Bilder, Texte und Empfehlungssysteme. Gegeben ist die nichtnegative Matrix $\boldsymbol{X}$ mit Dimension $m \times n$, die wir in die zwei Matrizen $\boldsymbol{W}$ und $\boldsymbol{H}$ mit den Dimensionen $m \times k$ und $k \times n$ zerlegen wollen, so dass

$$\boldsymbol{X} \approx \boldsymbol{W}\boldsymbol{H}.$$

wobei k die Anzahl der latenten Merkmale ist.

NMF | Rekonstruktionsfehler minimieren

Wir wollen $\boldsymbol{W}$ und $\boldsymbol{H}$ finden, so dass die Differenz zwischen $\boldsymbol{X}$ und $\boldsymbol{WH}$ minimiert wird. Die folgenden multiplikativen Aktualisierungsregeln minimieren den Rekonstruktionsfehler auf Basis der Frobenius-Norm [Lee & Seung, 2001]:

$$\min$$

NMF Schritt 1 | Nichtnegative Faktoren initialisieren

Eine gebräuchliche Methode dafür ist der folgende iterative Algorithmus.

Schritt 1 - Initialisiere: Initialisiere \boldsymbol{W} und \boldsymbol{H} mit zufälligen nichtnegativen Werten.

NMF Schritt 2 | Faktoren iterativ aktualisieren

Schritt 2 - Iterative Aktualisierung: Aktualisiere \boldsymbol{W} und \boldsymbol{H} iterativ unter Verwendung der multiplikativen Aktualisierungsregeln:

\boldsymbol{H} \leftarrow \boldsymbol{H} \odot \frac{\boldsymbol{W}^T \boldsymbol{X}}{\boldsymbol{W}^T \boldsymbol{W} \boldsymbol{H}}

\boldsymbol{W} \leftarrow \boldsymbol{W} \odot \frac{\boldsymbol{X} \boldsymbol{H}^T}{\boldsymbol{W} \boldsymbol{H} \boldsymbol{H}^T}

wobei \odot das elementweise Produkt und \frac{A}{B} die elementweise Division bezeichnen. Neben dieser Variante auf Basis der Frobenius-Norm gibt es auch andere Zielfunktionen, z.B. Divergenzmaße für Zähldaten.

NMF Schritt 3 | Abbruchkriterium prüfen

Schritt 3 - Konvergenzkriterium: Überprüfe die Konvergenz, z.B. ob die Änderung des Rekonstruktionsfehlers unter einen bestimmten Schwellenwert fällt oder eine maximale Anzahl von Iterationen erreicht ist.

NMF Schritt 4 | Ursprungsmatrix rekonstruieren

Schritt 4 - Rekonstruktion: Die rekonstruierte Matrix $\boldsymbol{\hat X}$ ist das Produkt der Faktormatrizen $\boldsymbol{W}$ und $\boldsymbol{H}$:

$$\boldsymbol{\hat X} = \boldsymbol{W} \boldsymbol{H}.$$

NMF in SciKit-Learn

Auch für die nichtnegative Matrixfaktorisierung gibt es eine Implementierung in `sklearn`. Zu beachten ist, dass sie allerdings nur nichtnegative Werte verarbeiten kann. Deshalb müssen wir unsere Daten zuerst auf einen positiven Wertebereich *normalisieren*. Dafür können wir den `MinMaxScaler` verwenden, der unsere Daten auf den Wertebereich $0,\ldots,1$ transformiert.

$$\boldsymbol X_\text{mx} =\frac{\boldsymbol X – \boldsymbol \min_X}{\boldsymbol \max_X - \boldsymbol \min_X}.$$

mit dem Vektor der minimalen Werte $\boldsymbol \min_X$ und maximalen Werte $\boldsymbol \max_X$ aller Variablen in $\boldsymbol X$.

```
from sklearn.preprocessing import MinMaxScaler

mms = MinMaxScaler()
egywthMinMax = mms.fit_transform(egywthNumber)
mms.min_, mms.scale_
```

```
(array([  0.        ,  -0.14705882,  -0.17349138,  -0.19689119,
          0.        ,  -0.14126394,  -0.08982036,   0.        ,
          0.        ,   0.        ,   0.        ,   0.        ,
         -0.13333333, -14.3641576 ,   0.1627907 ,  -0.55099483,
          0.09004739,   0.28315412,   0.415625  ]),
 array([0.00022624, 0.00022624, 0.00053879, 0.00259067, 0.00847458,
        0.03717472, 0.05988024, 0.02427184, 0.125     , 0.06198475,
        0.08333333, 0.125     , 0.05128205, 0.01470156, 0.02906977,
        0.01566661, 0.02369668, 0.03584229, 0.03125   ]))
```

NMF in sklearn | Modell auf die skalierten Daten fitten

Die Anwendung der NMF ist nun einfach.

```
from sklearn.decomposition import NMF

nmfm = NMF(n_components=2)
W = nmfm.fit_transform(egywthMinMax)
H = nmfm.components_
```

NMF in sklearn | Beobachtungen über W beschreiben

Hier gibt nun die Matrix W an, aus welchen Komponenten sich jede Beobachtung in unserem Datensatz zusammensetzt. Zum Beispiel sehen wir, dass sich die ersten zwei Zeilen nur aus der Komponente 1 zusammensetzen.

```
pd.DataFrame(W, columns=[f'Merkmal {i+1}' for i in range(n_components)])
```

	Merkmal 1	Merkmal 2
0	0.237477	0.000000
1	0.241596	0.000000
2	0.223237	0.016886
3	0.222422	0.028341
4	0.252235	0.000574
...	...	...
911	0.202307	0.057166
912	0.186173	0.083753
913	0.200195	0.037590
914	0.209245	0.025975
915	0.205023	0.028051

916 rows × 2 columns

NMF in sklearn | Komponenten über H interpretieren

Interessanter ist die Matrix H, weil sie die extrahierten Komponenten darstellt. Hierfür transformieren wir sie zurück in den originalen Datenbereich, indem wir den MinMaxScaler mit `inverse_transform` anwenden.

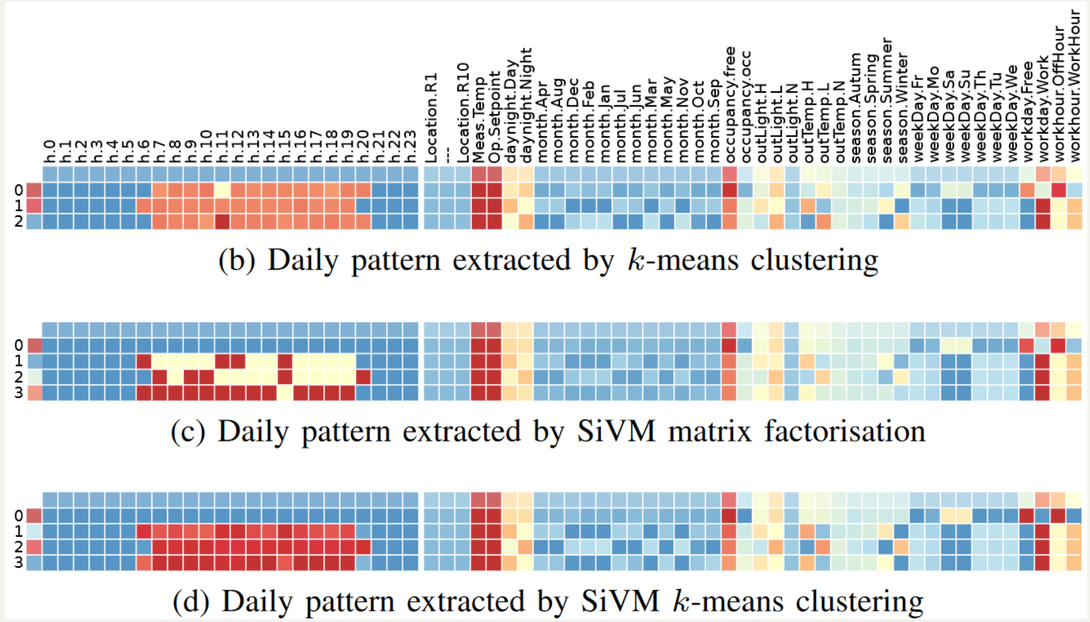
```
H_rescaled=mms.inverse_transform(H)
pd.DataFrame(H_rescaled, columns=egywthNumber.columns)
```

	EV_HT_740	EV_NT_740	E_AV_Lab	E_SV_Lab	ES_Lab	FX	FM	RSK	RSKF	SDK	SHK_TAG	NM	VPM	PM	TMK
0	5016.697406	10212.326793	3809.19813	881.823074	0.000000	45.343953	18.038292	12.541693	30.164496	0.000000	1.338495	33.113833	23.604693	1102.063064	36.801254
1	2445.902565	4540.833326	2136.22941	468.661514	199.960004	14.237872	5.890776	0.000000	0.000000	26.505994	0.000000	4.171526	25.429733	1063.255817	42.593383

Diese zwei Komponenten sind nun zwei charakteristische Kombinationen, mit denen der Datensatz in dieser niedrigdimensionalen Darstellung approximiert wird. So lassen sich aus großen Datensätzen wichtige Muster ableiten. Man kann sie mit Vorsicht ähnlich wie typische Muster oder Prototypen auffassen.

NMF-Anwendung | Gebäudebetriebsmuster vergleichen

Die folgende Abbildung aus der Publikation "Understanding building operation from semantic context" [Ploennigs et al., 2015] zeigt das an einem Beispiel. In der Publikation werden die Steuerungsstrategien von Gebäuden aus Verbrauchsdaten abgeleitet. Die Abbildung vergleicht die Muster, die durch Clustering (k-Means), SiVM-Matrixfaktorisierung (vergleichbar mit NMF, die insbesondere charakteristische Muster identifiziert) und SiVM-Clustering (eine Kombination beider) identifiziert werden. Hierbei zeigt sich, dass die Muster, die durch SiVM-Matrixfaktorisierung identifiziert werden, deutlich ausgeprägter sind. Das liegt an der besonderen Methode *SiVM* zur Identifikation der Komponenten. Daraus wurden anschließend über *Assoziationsanalyse* Betriebsregeln abgeleitet, die Entscheidungsbäumen ähneln.



Referenzen

Lee, D. D., & Seung, H. S. (2001). Algorithms for Non-negative Matrix Factorization. *Advances in Neural Information Processing Systems 13*.
Ploennigs, J., Chen, B., & Schumann, A. (2015). Understanding building operation from semantic context. *IECON - An. Conf. IEEE Ind. Elec. Soc.*

QUESTIONS