Efficient Event Classification through Constrained Subgraph Mining

Abschlussvortrag Bachelorarbeit

Simon Lackerbauer

2018-04-23

Outline

Outline

Problemstellung

- Datenherkunft: Industrielle Fertigungsstrecke von Siemens
- Datenart: Fehlermeldungen der verschiedenen Fertigungsmodule
- Daten sind proprietär, deshalb wurde ein zusätzliches, synthetisches Datenset konstruiert, das bei den meisten Folien zum Einsatz kommt

Problemstellung

- Die Anlage hat viele Ausfälle
- Ziel war es, Patterns aus den Daten zu generieren, von denen auf die Ursprünge der Probleme beim Ablauf geschlossen werden kann
- Mit diesen Patterns sollten die eigentlichen Anlagentechniker die Gründe der häufigen Ausfälle ausmachen und dementsprechend mitigieren können

Beispieldaten

Table 1: Synthetisches Datenset (Auszug)

time stamp	log message	module id	part id
2017-04-05 11:01:05	Laser überhitzt	Module 1	88495775TEST
2017-04-05 11:01:05	Laser überhitzt	Module 1	88495776TEST
2017-04-05 11:01:06	Teil verkantet	Module 2	88495776TEST
2017-04-05 11:01:06	Laser überhitzt	Module 1	88495776TEST
2017-04-05 11:01:10	Laser überhitzt	Module 1	88495776TEST
2017-04-05 11:01:12	Auffangbehälter leeren	Module 2	88495775TEST
2017-04-05 11:01:17	Unbekannter Ausnahmefehler	Module 0	88495775TEST
2017-04-05 11:01:17	Auffangbehälter leeren	Module 2	88495775TEST
2017-04-05 11:01:19	Unbekannter Ausnahmefehler	Module 0	88495775TEST
2017-04-05 11:05:22	Laser überhitzt	Module 1	88495775TEST
⋮	⋮	⋮	⋮

Problemstellung

Fehlermeldungen sind

- komplett unstrukturiert
- vollständig Deutsch
- sehr kurz, bzw. keine vollständigen Sätze
- teilweise nur für Experten verständlich

Evaluation

- Als weitere Metrik über die Anlage wurde die *Overall Equipment Efficiency* (OEE) bereitgestellt
- Der OEE-Score ist eine Größe zwischen 0 und 1, die sich folgendermaßen berechnet: $OEE = \frac{POK \cdot CT}{OT}$
- Auf der OEE-Zeitreihe wurde eine Anomalie-Detektion durchgeführt
- Die gefundenen Patterns sollten dann diese Anomalien vorhersagen

Outline

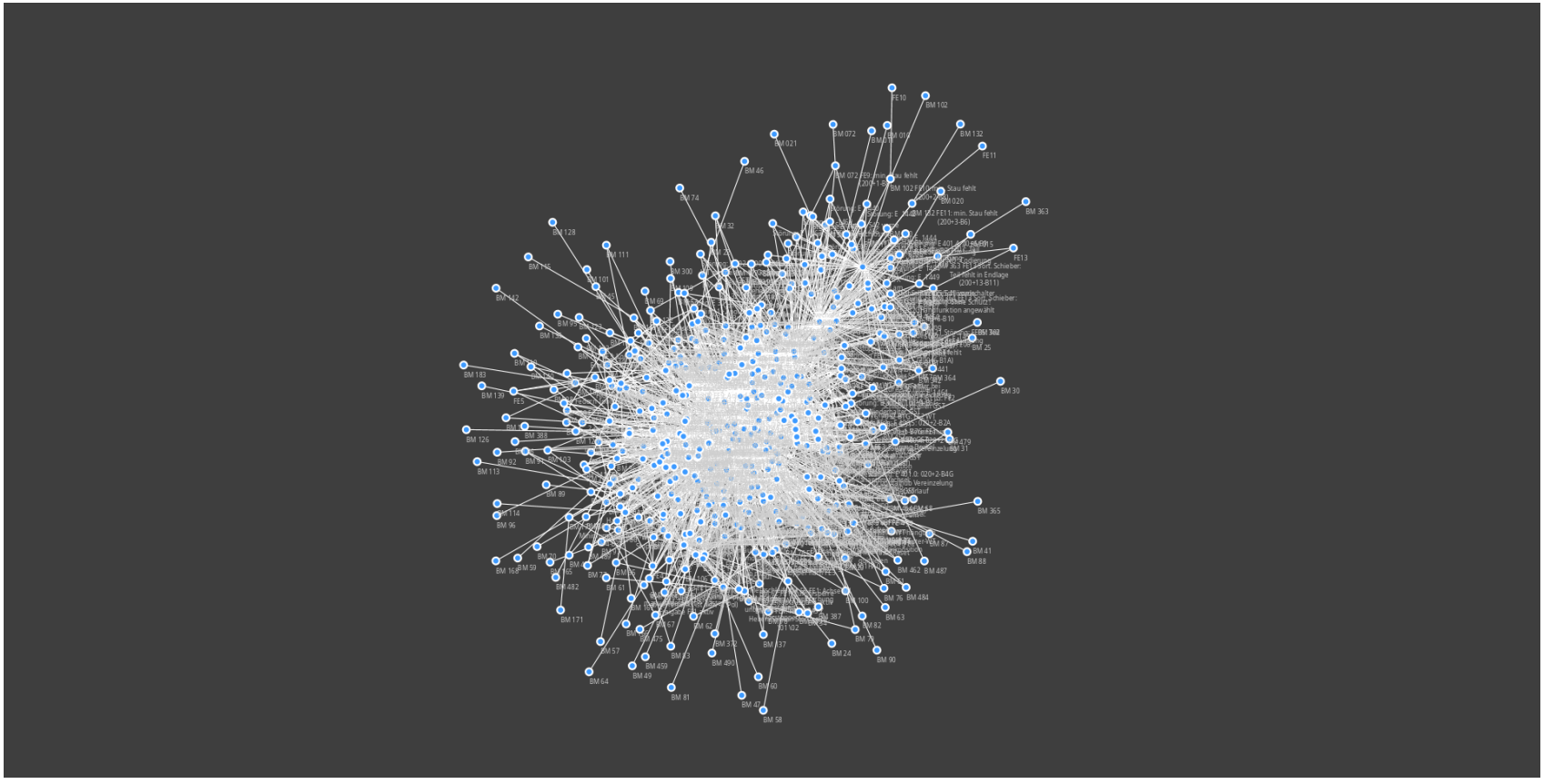
Sequenzpattern-Mining

- Sequenzen von *frequent patterns* zu generieren, führte bereits zu kleinen Erfolgen
- Die gefunden Patterns waren jedoch leider den Technikern mit Expertenwissen bereits bekannt

Erster Ansatz: Ein einzelner großer Graph

- Es gibt bereits Ansätze zum Mining von Patterns auf großen Graphen, (vgl. *GRAMI*, Elseidy et al, 2014, und *POSGRAMI*, Moussaoui et al, 2016)
- Eine eigene Idee war, mittels der Suche nach kürzesten Pfaden (Dijkstra), längere, aufeinander aufbauende, und damit vermutlich kausal zusammenhängende, Pfade zu finden

Darstellung großer Graph



Outline

Graph-Aufbau

- Die Daten wurden unter Verwendung von Wissen um den Anlagen-Aufbau als *constraints* in eine Graph-Form gebracht
- Jeder Graph enkodiert 5 Minuten an Informationen
- Auf der Menge der generierten Graphen wird dann der *gSpan*-Algorithmus zur Pattern-Suche ausgeführt

Graph-Isomorphismus

- Das Grundproblem beim Graph-Mining ist die Feststellung, ob zwei (Sub-)Graphen zueinander isomorph sind
- Def.: Seien G und H Graphen. Sei $f: V(G) \rightarrow V(H)$ eine Bijektion und $u, v \in V(G), (u, v) \in E(G)$. Dann gilt $G \simeq H$ g.d.w $(f(u), f(v)) \in E(H)$.
- Das Subgraph-Isomorphie-Problem ist NP-complete

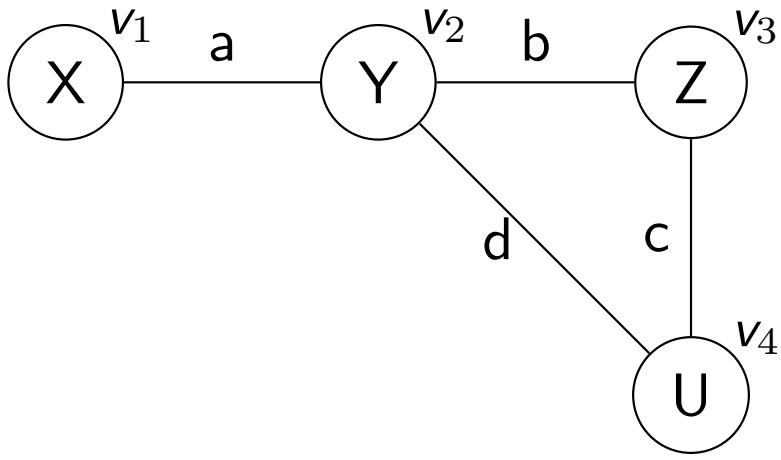
gSpan

- *gSpan* ist ein pattern-growth Algorithmus von *Yan und Han* aus 2002
- *gSpan* weist jedem Graph ein kanonisches, auf DFS traversal basierendes Label zu (DFS-Codes)
- Zwei Graphen mit gleichem Label sind isomorph
- *gSpan* findet sodann alle Subgraphen der Elemente einer Menge von Graphen, welche einen *minimum support threshold* (*min_sup*) erreichen.

Modifikation von gSpan

- Beim Implementieren von *gSpan* in Python fiel auf, dass die DFS-Codes ähnlich wie Hashes funktionieren, aber die verwendete Datenstruktur Vergleichsoperationen nicht sehr effizient macht
- Leider kann gSpan nicht vollständig auf den reinen Vergleich von Hashes umgestellt werden, da über der Menge der DFS-Codes eine starke Totalordnung liegen muss

Beispiel DFS-Code



edge no.	DFS code
0	$(0, 2, U, d, Y)$
1	$(1, 2, X, a, Y)$
2	$(0, 3, U, c, Z)$
3	$(2, 3, Y, b, Z)$

Pattern-growth Aspekt

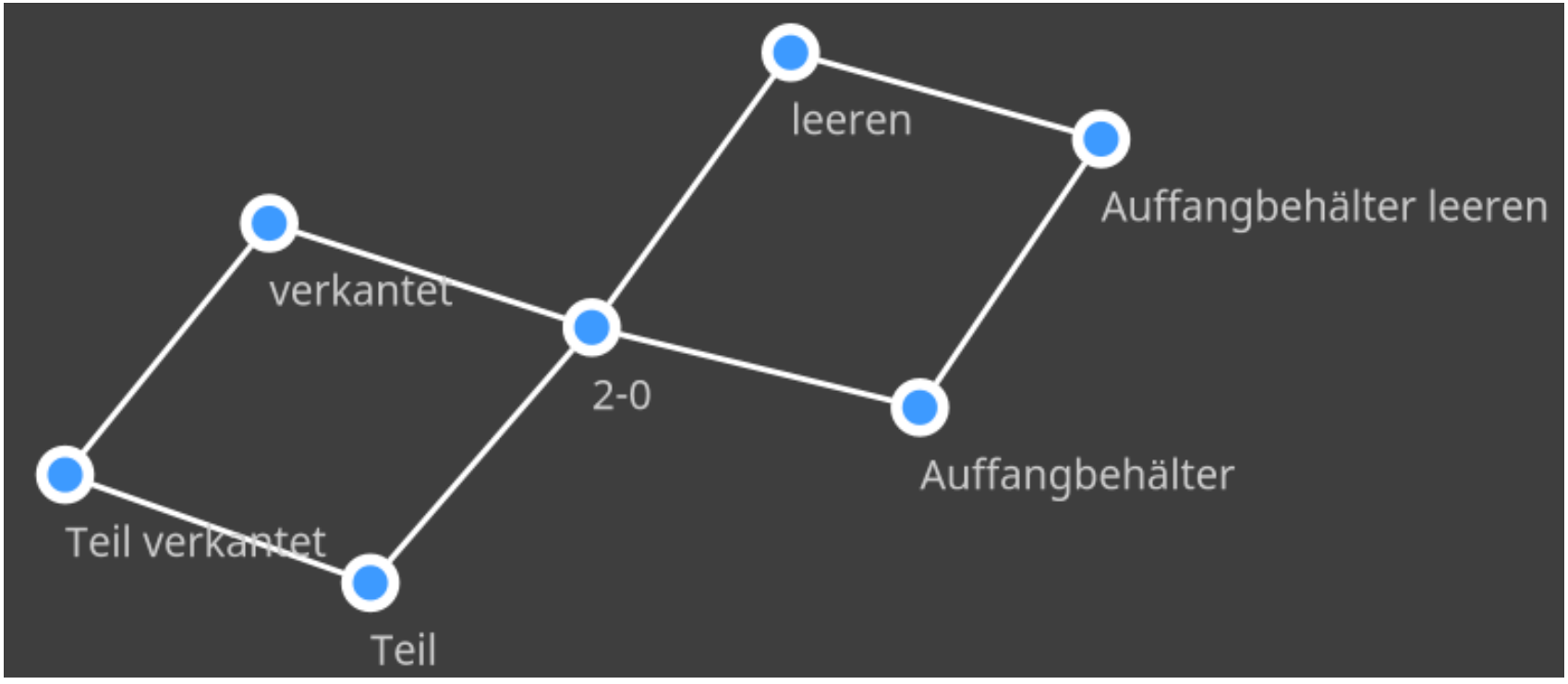
- Beim Suchen nach neuen Patterns verwendet *gSpan* die schon gefundenen Patterns
- Pattern-Kandidaten können neue Kanten nur am *rightmost path* anfügen, was den Suchraum eingrenzt

Support Vector Machine

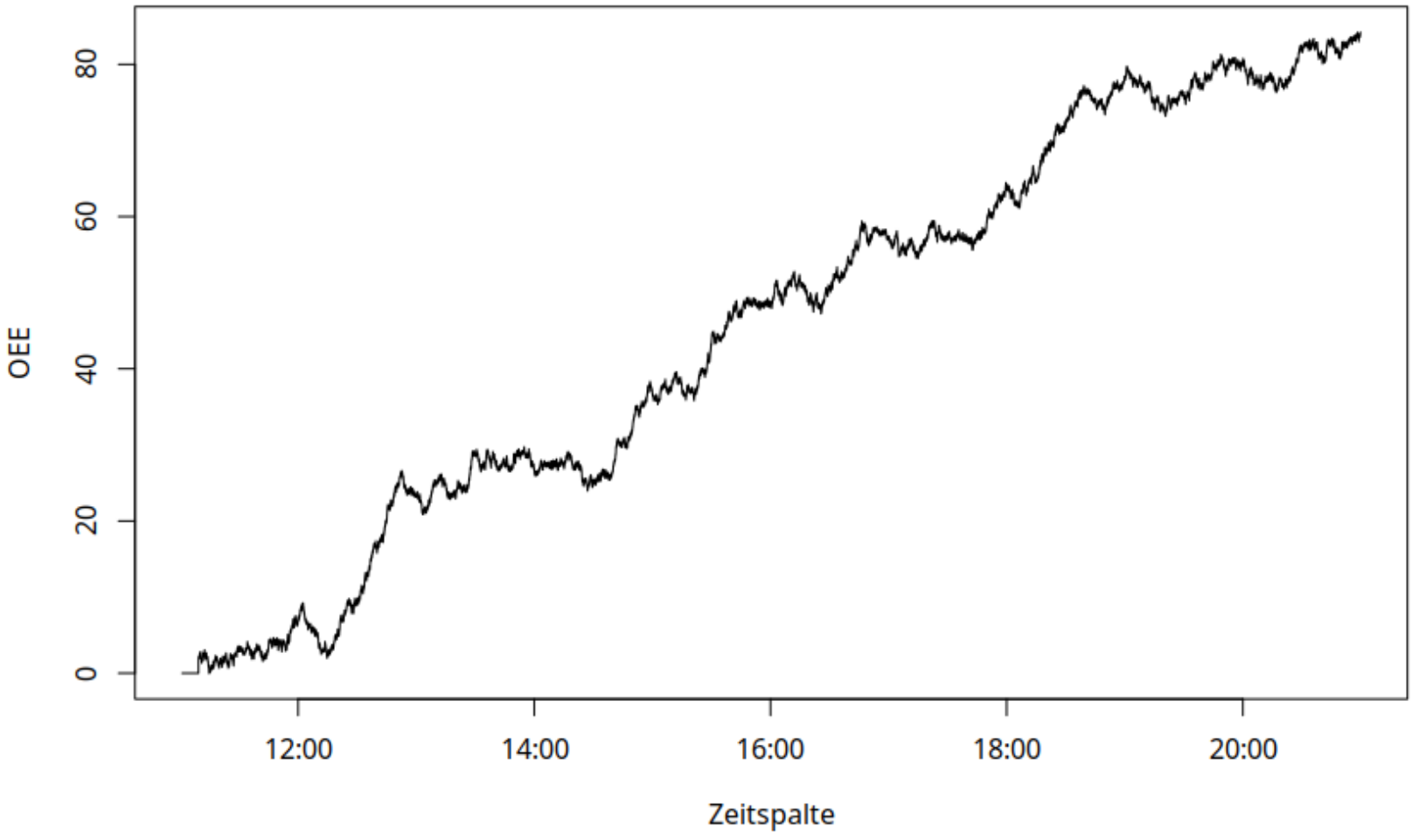
- Zum Klassifizieren der Patterns zu den gefundenen Anomalien wurde eine SVM eingesetzt
- Eine SVM ist ein supervised learning Modell, das relativ effizient hochdimensionale Datenpunkte auf zwei Klassen verteilen kann

Outline

Beispiel-Pattern



Synthetischer OEE-Verlauf



Laufzeiten synthetische Daten

data set	t	patterns
import errors and graph generation	1s	
import and anomalies detection on OEE scores	8s	
gSpan (min_sup = .7)	2s	40
gSpan (min_sup = .6)	8s	106
gSpan (min_sup = .5)	19s	241
gSpan (min_sup = .4)	74s	1056
SVM training and validation (min_sup = .7)	4s	
SVM training and validation (min_sup = .6)	8s	
SVM training and validation (min_sup = .5)	35s	
SVM training and validation (min_sup = .4)	13m 14s	

The validation data set consisted of 49 time windows, 33 of which were deemed as a noticeable drop by the OEE evaluation algorithm. Of these 33, the SVM correctly identified 28 as drops, for a sensitivity score of 84.85%. Of the remaining 19 non-drops, 5 were falsely identified as positives, for a specificity score of 73.68%.

Laufzeiten reale Anlagendaten

data set	t	patterns
import errors and graph generation	50s	
import and anomalies detection on OEE	2m 27s	
gSpan (min_sup = .9)	2m 20s	12
gSpan (min_sup = .7)	6h 27m 12s	846
gSpan (min_sup = .5)	*OOM killed*	–
SVM training and validation (min_sup = .7)	27s	

The validation data set consisted of 486 time windows, 64 of which were deemed as a noticeable drop by the OEE evaluation algorithm. Of these 64, the SVM trained on patterns with a min_sup of .7 correctly identified 60 as drops, for a sensitivity score of 93.75%. Of the remaining 422 non-drops, 18 were identified as false positives, for a specificity score of 95.73%.