INSTITUT FÜR INFORMATIK
Datenbanken und Informationssysteme

Universitätsstr. 1 D–40225 Düsseldorf

Heinrich Heine Universität Düsseldorf

Analyse des Kundenverhaltens durch Mustererkennung in Logfiles

Markus Stroh

Bachelorarbeit

Beginn der Arbeit:	18. Februar 2020
Abgabe der Arbeit:	18. Mai 2020
Gutachter:	Prof. Dr. Stefan Conrad
	Prof. Dr. Michael Leuschel

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig verfasst habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Düsseldorf, den 18. Mai 2020

Markus Stroh

Zusammenfassung

Die vorliegende Arbeit, die am Lehrstuhl für Datenbanken und Informationssysteme der Heinrich-Heine-Universtät Düsseldorf verfasst wurde, beschäftigt sich mit der Mustererkennung in Logfiles. Dabei soll das Kundenverhalten des Produkts *Multiversa International Finance Portal* der Firma Computer-Communications Networks GmbH erkannt werden. Dafür wurden zunächst die Firma und das Produkt vorgestellt. Die Analyse des Kundenverhaltens soll auf zwei Arten erfolgen. Zum einen war eine visuelle Darstellung erwünscht, zum anderen wurde ein Verfahren zur Suche nach Assoziationsregeln gesucht. Mit letzterer Methode sollen Workflows des Kunden erkannt werden.

Das System, das im Rahmen dieser Arbeit entwickelt wurde, basiert auf den Produkten Elasticsearch, Kibana, Logstash und Filebeat der Firma Elasticsearch N.V. Im Verlauf der Arbeit wird dargestellt, wie diese Produkte eingesetzt wurden und ggf. um eigene Funktionen erweitert wurde. Insbesondere im Bezug auf die Suche nach Assoziationsregeln mussten diese Produkte um Funktionen erweitert werden, die die Software nativ nicht anbietet. Da die Nutzerfreundlichkeit bei der Arbeit nicht zu vernachlässigen war, wurde ein Custom Plugin für Kibana entwickelt, das Python Skripte ausführen kann, mit denen die Assoziationsregeln gefunden wurden. Dadurch sind keine Vorkenntnisse nötig, um als User das Analyse Tool zu benutzen.

Nachdem die theoretischen und technischen Grundlagen geschaffen wurden, wurde das System getestet. Dabei wurden die Vorgehensweise und die Auswertung dokumentiert. Das Ergebnis der Arbeit ist neben dem entwickelten System eine kritische Auseinandersetzung mit den Ergebnissen der Tests und der angewandten Methoden. Den Abschluss bildet eine Aussicht darauf, wie das System für zukünftige Anwendungsfälle optimiert bzw. erweitert werden kann.

Inhaltsverzeichnis

1 Einleitung

In dem ersten Kapitel der Arbeit wollen wir die Problematik und die Aufgabenstellung definieren. Dazu wird zunächst das Unternehmen Computer-Communications Networks GmbH (CoCoNet) vorgestellt. Nachdem die Arbeit dadurch in den Firmenkontext gebracht wird, wollen wir konkret die Fragestellung der Arbeit definieren und deren Aufbau beschreiben.

1.1 Firmenprofil CoCoNet GmbH

Die Firma CoCoNet entwickelt digitale Banking-Lösungen für Unternehmen. In dieser Arbeit beschäftigen wir uns mit einem bestimmten CoCoNet Produkt, das *Multiversa International Finance Portal* (IFP). Das IFP ist eine in Java implementierte Webanwendung, die alle gängigen Browser unterstützt. Nachdem sich ein User in das IFP eingeloggt hat, wird er zunächst zum Dashboard weitergeleitet. Das Dashboard ist die Startseite des IFPs, bestehend aus einer Navigationsleiste und Dashboardwidgets. Ein Widget ist eine Einheit, die auf dem Dashboard abgelegt werden kann und eine bestimmte Funktionalität hat. Von hier aus haben User die Möglichkeit durch das IFP zu navigieren, um z.B. eine Überweisung zu tätigen. (CoCoNet, 2020)
Für den weiteren Verlauf der Arbeit wollen wir nicht den kompletten Funktionsumfang des IFPs beleuchten, sondern uns auf das Dashboard mit seinen Widgets fokussieren. Betrachten wir als Beispiel das *Open Payments Widget* in Abbildung 1.

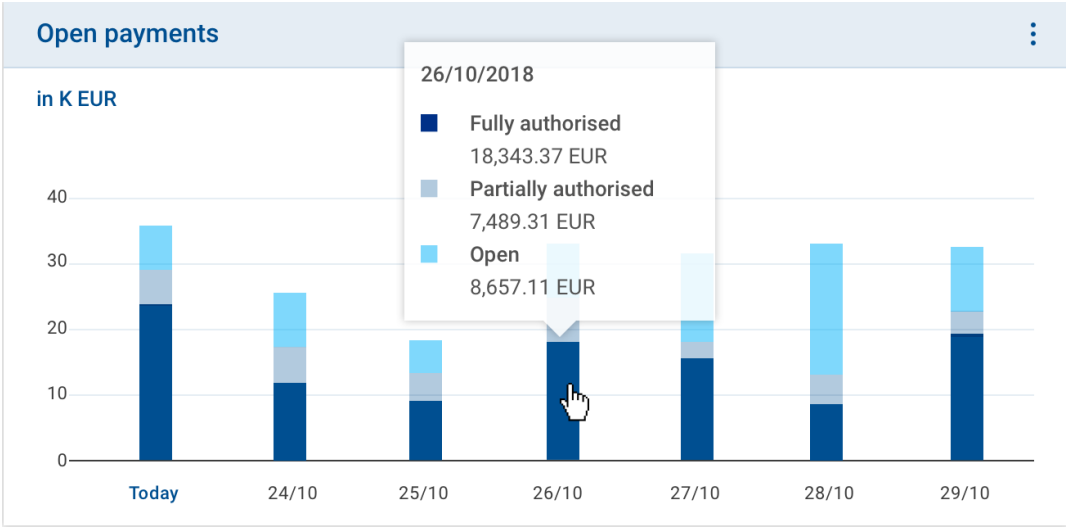

Abbildung 1: Open Payments Widget

Wie der Name schon vermuten lässt, zeigt das Widget Zahlungen an, die noch offen sind, also authorisiert werden müssen. Klickt man nun eine offene Zahlung an, wird man auf eine entsprechende Seite weitergeleitet, die mit den Zahlungsinformationen der selektierten Zahlung ausgefüllt ist.

1.2 Ziel der Arbeit

In dieser Arbeit geht es darum, das Userverhalten in Hinblick auf die Nutzung der Dashboardwidgets im IFP zu analysieren. Da der Begriff Userverhalten noch recht breit gefächert ist, sollen konkret die folgenden Fragen beantwortet werden:

1. Wie oft werden Widgets in einem bestimmten Zeitraum benutzt?

2. Lässt sich ein bestimmter Workflow durch die Nutzung der Widgets erkennen?

Diese Fragen wollen wir durch die Analyse von Logfiles beantworten.
Zunächst müssen wir aber definieren, was es bedeutet, ein Widgets zu benutzen. Da das Dashboard bzw. die Widgets das Erste sind, was man nach einem Login in das IFP sieht, könnte man meinen, dass das Vorhandensein auf dem Dashboard eine Nutzung schon einschließt. Betrachtet man z.B. das Widget in Abbildung 1 stellt man fest, dass das Widget schon einiges an Informationen liefert. Allerdings kann man an dieser Stelle nur spekulieren, ob und in welchem Ausmaß der User diese Information wahrnimmt. Ein pures Vorhandensein des Widgets auf dem Dashboard ist als Nutzung nicht hinreichend. Erst ein Klick auf das Widget zählt als Nutzung.
Wenn nun ein Widget angeklickt wird, wird der Nutzer in der Regel auf eine neue Seite weitergeleitet, die durch das Widget z.B. schon vorgefiltert ist. Da diese Weiterleitung ein HTTP-Request an den Server ist, wird sie geloggt und kann in den Logfiles erkannt werden.
Also sind die Anforderungen an das zu entwickelnde System, dass die Nutzung eines Widgets erkannt wird und aus diesen Informationen Verhaltensmuster der User bzgl. der eben erwähnten Fragen erkannt werden. Außerdem soll das System nutzerfreundlich sein, damit es auch Menschen ohne besondere Vorkenntnisse benutzen können.
Ein weiteres Ziel der Arbeit ist eine kritische Einschätzung, inwiefern sich das entwickelte System für diese Art der Kundenanalyse eignet.

1.3 Aufbau der Arbeit

Im folgenden Verlauf der Arbeit wird zunächst auf die theoretischen Grundlagen der Mustererkennung eingegangen. Dazu wird die Syntax der vorliegenden Logfile Einträge und anschließend die Phasen der Mustererkennung, die die Einträge durchlaufen, beschrieben. In dem darauf folgenden Kapitel wird dargelegt, wie die theoretischen Grundlagen technisch umgesetzt werden. Danach wird das entwickelte System getestet und die Ergebnisse diskutiert. Zum Schluss wird die Arbeit durch ein Fazit zusammengefasst und zukünftige Einsatzmöglichkeiten erörtert.

2 Grundlagen

In diesem Kapitel beschäftigen wir uns mit den theoretischen Grundlagen der Methoden, die benutzt wurden, um die Kundendaten zu analysieren. Dafür wollen wir zunächst die Syntax der Logfiles beschreiben. Das dient als Vorbereitung für den Abschnitt über die Mustererkennung. In diesem werden die verschiedenen Phasen der Mustererkennung beschrieben sowie die Methoden, die im weiteren Verlauf der Arbeit angewendet werden.

2.1 Syntax der Logeinträge

Das IFP generiert bei seiner Benutzung mehrere verschiedene Logfiles. Je nach Art des Logfiles können hier verschiedene Informationen gespeichert werden wie z.B. Zeitstempel, UserIDs, Exception Messages usw. Da für diese Arbeit aber nur die sog. *Sessionlogs* relevant sind, soll auch nur deren Syntax beschrieben werden.
Die Einträge in den Sessionlogs spiegeln im Großen und Ganzen die Serveraktivität wider: Es werden ein- und ausgehende HTTP Requests festgehalten. Allerdings ist es nicht ausreichend nur eine Liste von Requests zu speichern. Sie müssen noch in einen gewissen Kontext gebracht werden. Deshalb werden zusätzlich zu den Requests auch Daten wie Zeitstempel, UserID, Parameter u.s.w. geloggt. Durch diese Informationen kann man nun nachvollziehen, welcher User zu welchem Zeitpunkt Daten gesendet bzw. angefragt hat. Neben der UserID wird auch die Session ID mit geloggt. Sobald sich ein User erfolgreich in das IFP einloggt, wird eine SessionID generiert. Diese SessionID ist vom Zeitpunkt des Logins bis zum Logout gültig. Möchte sich derselbe User nach dem Logout direkt noch einmal einloggen, wird eine neue Session ID generiert.
Die Einträge in den Logfiles können grob in die folgenden Felder unterteilen werden:

```
[Timestamp] [Loglevel] [SessionID] [CustomerID] [UserID] [TaskID] [HTTP Request]
```

Listing 1: Felder in Sessionlogs

Da nun die Syntax der Einträge erkannt wurde, kann die Mustererkennung beginnen.

2.2 Mustererkennung

Nach Beierle und Kern-Isberner (2019) lässt sich die Mustererkennung auch *Data Mining* nennen. Data Mining ist wiederrum Teil eines Prozesses, der sich *Knowledge Discovery in Databases* (KDD) nennt. Diesen Prozess haben Beierle und Kern-Isberner (2019) in acht Schritte aufgeteilt.
Allgemeiner lässt sich aber die Mustererkennung auch auf andere Bereiche anwenden. Beispiele dafür wäre die Mustererkennung in Bilddateien (Duda et al., 2001). Da wir durch die Einleitung in Kapitel 1 bereits eine Schritte des KDD-Prozesses abgedeckt haben, werden im weiteren Verlauf der Arbeit die Phasen nach Wang (2011) beschrieben. Der Vollständigkeit halber werden die Phasen nach Beierle und Kern-Isberner (2019) in

Klammern aufgeführt. Da die Aufteilung nach Wang (2011) allerdings kompakter ist, wird diese im weiteren Verlauf angewendet.

1. Segmentiereung (Datenauswahl / Datenbereinigung)
2. Feature Extraction (Datenreduktion und -projektion)
3. Klassifizierung (Modellfunktionalität / Verfahrensauswahl / Data Mining)

2.2.1 Segmentierung

In dem Schritt der Segmentierung wird festgestellt, welche Informationen aus den Logfiles für die Fragestellung relevant sind (Duda et al., 2001). Greifen wir dafür zunächst einmal die Definition aus der Einleitung auf, was es bedeutet ein Widget zu benutzen. Wie bereits erwähnt reicht ein pures Anzeigen von Informationen nicht aus; das Widget muss auch angeklickt werden. Ein Klick auf ein Widget hat i.d.R. immer denselben Effekt: man wird auf eine (evtl. vorgefilterte) Seite weitergeleitet. Das heißt, man sendet einen HTTP Request an den Server und der Server sendet eine Antwort. Also ist es offensichtlich, dass wir für unsere Analyse nur die HTTP Requests betrachten müssen und die Antworten des Servers irrelevant sind. Demnach sind nur Einträge von Interesse, die in dem Feld [HTTP Request] aus Listing 1 mit *Incoming Request* anfangen. Dadurch wird die Menge an Einträgen, die wir betrachten wollen, um ca. die Hälfte reduziert.
Betrachten wir nun den Rest des Feldes genauer anhand eines Beispieleintrags, in dem ein Widget benutzt wurde. Insbesondere die URL ist für uns von Interesse:

```
/MULTIVERSA-IFP/lightning/ecm/liquidity/banks/liquidity\_banks.
    jsf?selectedView=ecm.liquidity.banks.view.all\&viewType=0\&
    widget=LiquidityByBanksWidgetContent\&banks=
    RpBLVhy85c5u4nGKeISH0A\&conversationContext=3\&\_ns\_=
    f63a91bc-9ab1-472d-bcb0-b6771837ffaa3\&\_nc\_=
```

Listing 2: Beispiel URL

Der Name des Widgets ist in der URL erkennbar, wie man hier rot markiert sehen kann. Neben dem HTTP Request sind diese weiteren Felder von Relevanz:

- Timestamp
- SessionID
- UserID

2.2.2 Feature Extraction

Nachdem alle irrelevanten Daten in der Segmentierung eliminiert wurden, ist der nächste Schritt die Feature Extraction. Dieser Begriff könnte irreführend wirken, da in dieser

Arbeit keine Features im wörtlichen Sinne extrahiert werden, vielmehr werden die Daten in eine bestimmte Struktur transformiert (Beierle und Kern-Isberner, 2019). Diese besteht aus drei Mengen, die Menge an Usern, die Menge an SessionIDs und die Menge an Widgets. Von nun an wird diese Struktur *Session Entity* genannt. Zu einem User gehört eine Menge an SessionIDs. Diese ist verknüpft mit den Widgets, die in dieser Session benutzt wurden. Mit anderen Worten beinhaltet eine Session Entity die Informationen, welche Widgets in einer Session von wem benutzt wurden. Abbildung 2 soll diesen Zusammenhang verdeutlichen.

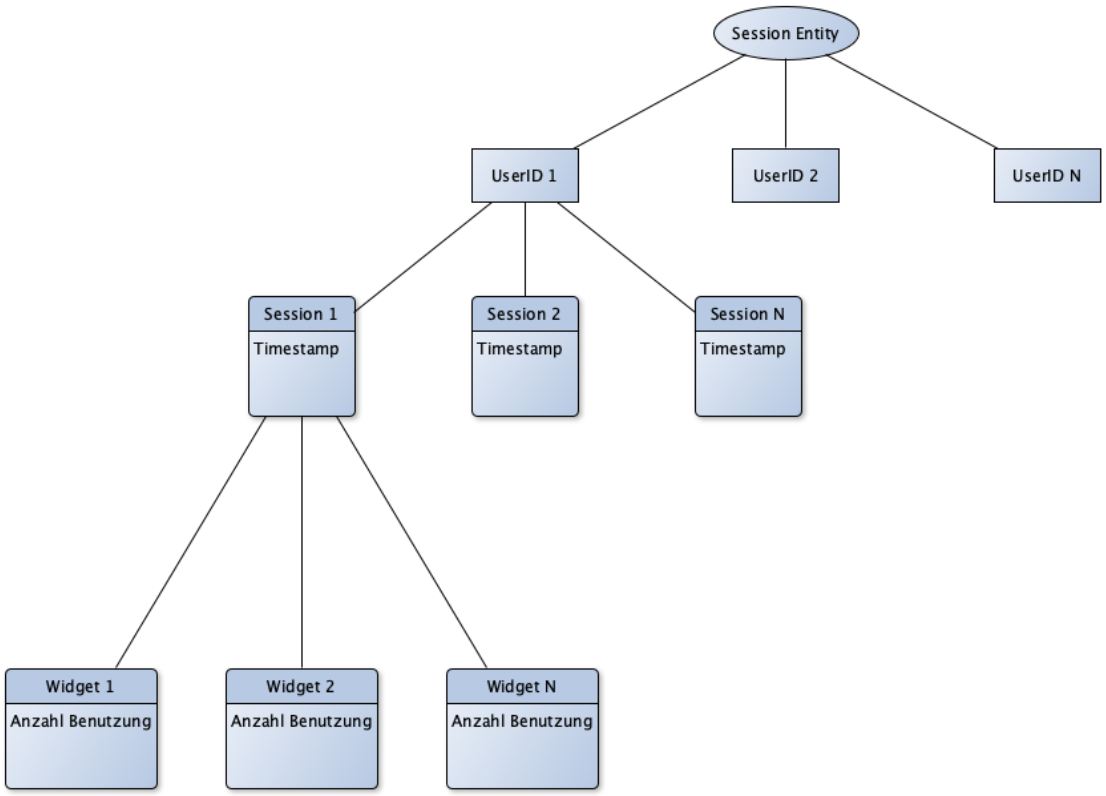

Abbildung 2: Session Entity

Zusätzlich zu der Information, welche Widgets benutzt wurden, wird noch festgehalten, wann eine Session beginnt und endet. Sowohl die Uhrzeit als auch die UserID werden im weiteren Verlauf der Arbeit keine tragende Rolle mehr spielen. Allerdings macht es Sinn diese Daten mitzuspeichern, da sie für zukünftige Anwendungsfälle relevant sein könnten.

2.2.3 Klassifizierung

Nachdem die Daten transformiert sind, werden sie im nächsten Schritt analysiert. In Abschnitt 1.2 wurde bereits erwähnt, dass in dieser Arbeit die Frage nach dem Zeitraum

der Nutzung der Widgets und dem Workflow zu erörtern sind. Deshalb wird das weitere Vorgehen für beide Fragen gesondert dargestellt.

Diagramm

Um die Frage zu klären, wie sich die Widgetnutzung pro Tag verhält, wird ein Linienhistogramm erstellt. Das hat den Vorteil, dass nicht nur die Nutzung an sich dargestellt wird, sondern auch ein Verlauf bzw. ein Trend in der Nutzung der Widgets. Zusätzlich wird in einem Tortendiagramm die anteilige Widgetnutzung dargestellt.

Assoziationsregeln

In dem folgenden Abschnitt werden die theoretischen Grundlagen zur Bestimmung von Assoziationsregeln vorgestellt. Zunächst sei aber drauf hingewiesen, dass dieser Abschnitt zu großen Teilen auf den Ausführungen von Ester und Sander (2000) in „*Knowledge Discovery in Databases*“ basiert. Andere Quellen, die zur Erstellung des Abschnitts benutzt wurden, sind entsprechend gekennzeichnet.
Die Suche nach Assoziationsregeln wird auch oft als Warenkorbanalyse bezeichnet. Dabei geht es darum, in einer Datenbank Beziehungen zwischen den Daten zu finden. Bevor wir in die Theorie hinter den Assoziationsregeln eintauchen, wollen wir mit Hilfe der folgenden Tabelle die wichtigsten Begriffe vorstellen. Zum besseren Verständnis zeigen wir zu den Begriffen, die eingeführt werden, Analogien zu unserem Anwendungsfall und eines Online Shops.

Allgemein	Online Shop	IFP
Transaktion	Ein einzelner Warenkorb	Eine einzelne SessionID
Menge aller Transaktionen D	Menge aller Warenkörbe D	Menge aller SessionIDs D
Item	ein Produkt	ein Widget bzw. eine Widgetnutzung
Menge aller Items I	Menge aller Produkte I	Menge aller Widgets I

Tabelle 1: Begriffe zu Assoziationsregeln

Nun wollen wir definieren was eine Assoziationsregel ist. Betrachten wir dafür die Widgets w_1 und w_2. Eine Assoziationsregel wird mit

$$w_1 \rightarrow w_2$$

notiert und sagt aus, dass wenn w_1 benutzt (also angeklickt) wird, w_2 auch benutzt wird. An eine Assoziationsregel sind zwei Werte geknüpft, die beschreiben, wie „gut“ eine Regel ist: der *Support* und die *Konfidenz*. Diese Werte wollen wir mit Hilfe eines kurzen Beispiels definieren:

SessionID	Benutzte Widgets
1	w_1, w_2, w_4, w_5
2	w_1, w_3
3	w_2, w_5
4	w_1, w_2
5	w_2, w_3, w_5

Tabelle 2: Beispiel Datenbank zu Assoziationsregeln

Zunächst werden die Begriffe aus Tabelle 1 auf das Beispiel der Datenbank (vgl. Tabelle 2) angewendet:

- Die Menge aller Items $I = \{w_1, w_2, w_3, w_4, w_5\}$. Eine Menge $X \subseteq I$ wird auch *Itemset* genannt. Also wäre z.B. $\{w_1, w_3\} \subseteq I$ ein Itemset.
- Die Menge aller SessionIDs $D = \{1, 2, 3, 4, 5\}$. T_i mit $i \in D$ und $T_i \subseteq I$. Demnach gilt z.B. $T_1 = \{w_1, w_2, w_4, w_5\}$

Der Support einer Menge wird definiert als:

$$\text{support(X)} = \frac{\text{Anzahl der Transaktionen, die X enthalten}}{\text{Anzahl aller Transaktionen}},\ \text{X} \subseteq I$$

Demnach gilt z.B. für $X_1 = \{w_1, w_3\}$, $X_2 = \{w_2\}$ und $X_3 = \{w_1\}$:

$$\text{support}(X_1) = \frac{1}{5} = 0.2,\ \text{support}(X_2) = \frac{4}{5} = 0.8,\ \text{support}(X_3) = \frac{3}{5} = 0.6$$

Der Support eines Itemsets beschreibt demnach die relative Häufigkeit des Itemsets, im Bezug auf die Anzahl der Transaktionen in der Datenbank (Beierle und Kern-Isberner, 2019). Desweiteren gilt für $X,Y \subseteq I$:

$$\text{support}(X \rightarrow Y) = \text{support}(X \cup Y)$$

Als nächstes wollen wir die Konfidenz definieren. Die Konfidenz einer Assoziationsregel $X \rightarrow Y$ ist definiert als:

$$\text{konfidenz}(X \rightarrow Y) = \frac{\text{support}(X \rightarrow Y)}{\text{support}(X)}$$

Auf Tabelle 2 bezogen wollen wir nun die Konfidenz für die Regel $\{w_1\} \rightarrow \{w_3\}$ berechnen. Es gilt:

$$\begin{aligned}\text{konfidenz}(\{w_1\} \rightarrow \{w_3\}) &= \frac{\text{support}(\{w_1\} \rightarrow \{w_3\})}{\text{support}(\{w_1\})} \\ &= \frac{\text{support}(\{w_1\} \cup \{w_3\})}{\text{support}(\{w_1\})} \\ &= \frac{0.2}{0.6} = 0.33\end{aligned}$$

Die Konfidenz beschreibt also wie hoch die Wahrscheinlichkeit ist, dass wenn w_1 benutzt w_3 auch benutzt wurde. (Beierle und Kern-Isberner, 2019)

Nachdem wir alle relevanten Begriffe bzgl. der Assoziationsregeln definiert haben, wollen wir genauer erörtern, wie Regeln gefunden werden sollen. Dazu rufen wir uns noch einmal ins Gedächnis, welche Information wir uns aus den Regeln erhoffen: wir wollen Workflows erkennen. Nehmen wir mal an, das Itemset $\{w_1, w_3\}$ käme in einer Datenbank mit 1000 Einträgen nur ein einziges Mal vor, ist es offentsichtlich, dass man bei dieser Regel nicht von einem Workflow sprechen kann. Das bedeutet wir suchen Itemsets, die *häufig* vorkommen. Diese Häufigkeit wird durch den *minsupport* festgelegt. Ein Itemset gilt als häufig, wenn gilt:

$$\text{support}(X) \geq \text{minsupport}$$

Analog wollen wir auch nicht alle Regeln, die wir aus den häufigen Itemsets generieren, betrachten. Vielmehr wollen wir die Regeln finden, die die *minkonfidenz* erfüllen. Daraus folgt, dass man die Aufgabe in zwei Teilprobleme aufteilen kann:

1. Finde alle Itemsets die häufig sind, also den minsupport erfüllen. Ein naiver Ansatz wäre, die Potenzmenge $\mathcal{P}(I)$ zu berechnen und zu prüfen, welche der Teilmengen den minsupport erfüllen. Allerdings hätte dieses Vorgehen eine Laufzeit von $\mathcal{O}(2^n)$ wobei n die Anzahl der Widgets ist. Deshalb wird der sog. *Apriori Algorithmus* verwendet, der die Laufzeit stark einschränkt, indem die zu betrachtenden Itemsets eingeschränkt werden.

2. Finde alle Assoziationsregeln, die aus den häufigen Itemsets generiert werden können und die die minkonfidenz erfüllen. Wenn Itemset X häufig ist, ist auch jede Teilmenge A von X häufig und ergibt die Regel A $\rightarrow$ X $-$ A. Für diese Regeln muss die Konfidenz geprüft werden.

Bevor wir nun die Algorithmen vorstellen, die zur Lösung der Teilprobleme benutzt wurden, sei noch angemerkt, dass der minsupport und die minkonfidenz Werte sind, die vom Benutzer festgelegt sind.

Apriori Algorithmus

Der Apriori Algorithmus wurde von Agrawal et al. (1993) entwickelt und wird benutzt, um möglichst effizient häufige Itemsets zu finden. Dabei wird folgendes Lemma ausgenutzt:

Lemma 1. *Sei X ein Itemset. Wenn X kein häufiges Itemset ist, ist auch keine Obermenge von X häufig. (Agrawal et al., 1993)*

Ausgangspunkt der Kandidatengenerierung ist, dass man häufige Itemsets mit k Elementen betrachtet. Aus diesen Itemsets sollen nun Itemsets generiert werden, die $k+1$ Elemente beinhalten. Um das zu erreichen, werden zwei Itemsets vereinigt, die $k-1$ gleiche Elemente haben. Angenommen, wir hätten die häufigen Itemsets $X = \{w_1, w_2, w_3\}$ und $Y = \{w_1, w_2, w_4\}$. Beide Mengen haben $k = 3$ Elemente. Um aus diesen Mengen eine $k+1 = 4$ - elementige Menge zu generieren, müssen sie $k-1 = 2$ gleiche Elemente haben. Da das der Fall ist, erhalten wir als Ergebnis die Vereinigung beider Mengen mit $X \cup Y = \{w_1, w_2, w_3, w_4\}$. Dieser Schritt wird *join* genannt.
Nach dem join muss noch geprüft werden, ob der generierte Kandidat auch Lemma 1 erfüllt. Dies geschieht beim *pruning* indem geprüft wird, ob jede $k-1$ - elementige Teilmenge des generierten Kandidaten häufig ist. Denn wenn eine Teilmenge des generierten Kandidaten nicht häufig ist, kann dieser auch nicht häufig sein und muss nicht weiter betrachtet werden. In Anhang Teil A wird die Kandidatengenerierung mit dem Beispiel aus Tabelle 2 vorgerechnet.

Assoziationsregeln finden

Aus den häufigen Itemsets, die der Apriori Algorithmus gefunden hat, wollen wir nun Assoziationsregeln finden. Ähnlich wie bei den häufigen Itemsets wollen wir nur

Regeln finden, die die minkonfidenz erfüllen. Das erreichen wir, indem aus einem häufigen Itemset X zunächst Regeln gebildet werden, die der Form

$$Y \rightarrow X - Y, Y \subset X$$

entsprechen. Ähnlich wie bei der Kandidatengenerierung nutzen wir wieder eine Teilmengenbeziehung aus: Wenn die Regel $Y \rightarrow X - Y$ die minkonfidenz nicht erfüllt, erfüllt auch keine Teilmenge $Y' \subset Y$ mit der Regel $Y' \rightarrow X - Y'$ die minkonfidenz.

3 Umsetzung

Im Anschluss an Kapitel 2, in dem die theoretischen Grundlagen erläutert wurden, beschreibt dieses Kapitel, wie die Grundlagen umgesetzt werden. Zuerst wird die Funktionsweise des Elastic Stacks erklärt. Dabei werden seine Komponenten schrittweise vorgestellt. Zum Abschluss des Kapitels wird auf die Implementierung des Apriori Algorithmus eingegangen.

3.1 Grundlagen zum Elastic Stack

Um die Fragen aus der Problemstellung zu beantworten, wurde der Elastic Stack (ELK) benutzt. Der ELK besteht aus den Software Produkten Elasticsearch, Logstash und Kibana der Firma Elastic N.V. Zusätzlich wurde noch Filebeat von der gleichen Firma benutzt. Die Funktionen der einzelnen Produkte werden in den nachfolgenden Unterkapiteln weiter erläutert. Die folgende Abbildung stellt den Workflow des Systems dar:

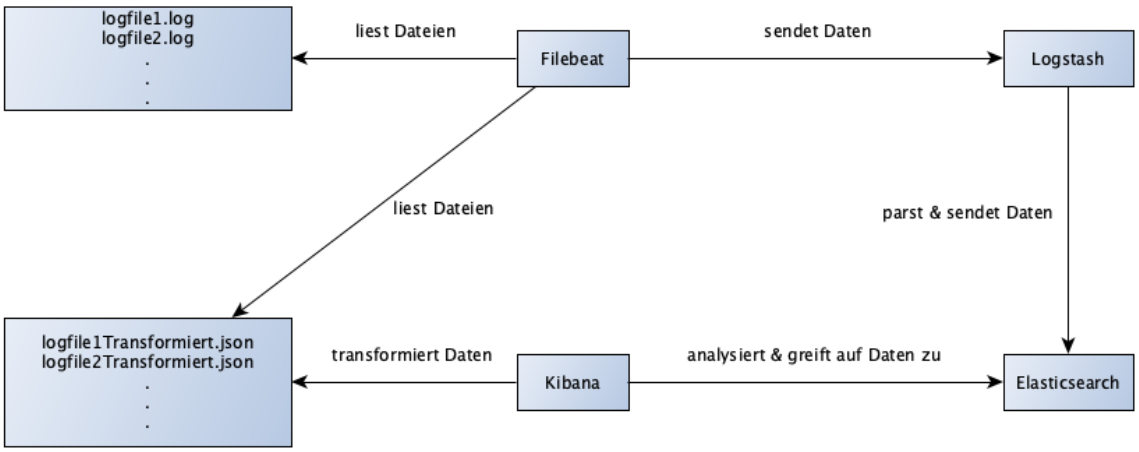

Abbildung 3: ELK Workflow

3.1.1 Filebeat

Mit Filebeat können Dateien zeilenweise gelesen werden. In der Konfigurationsdatei von Filebeat gibt man an wo und nach welchen Dateiendungen gesucht werden soll. In unserem Fall waren es .log und .json Dateien. Außerdem kann man ein multiline pattern angeben, falls ein Eintrag mehrzeilig sein sollte. Die eingelesenen Dokumente sendet Filebeat an Logstash weiter. Dabei merkt sich Filebeat, welche Dateien schon gelesen wurden und wie weit diese gelesen wurden. Das bedeutet ebenfalls, wenn eine komplett neue Datei oder eine neue Zeile in einer alten Datei auftaucht, merkt Filebeat das, liest es und sendet die Daten wieder an Logstash (Shukla und Sharath, 2017).

3.1.2 Logstash

Die Aufgabe von Logstash ist Daten von Filebeat zu empfangen und zu verarbeiten. Die empfangenen Daten befinden sich in einem rohen Zustand und werden mit Hilfe von mehreren Filter-Plugins verarbeitet (Elastic, 2020j). Nachdem die Daten verarbeitet wurden, sendet Logstash sie weiter an Elasticsearch.

Grok Filter-Plugin

Für die Logdateien wird das Grok Filter-Plugin verwendet. In dem Grok Filter gibt man einen regulären Ausdruck an, mit dem ein Eintrag in den Logfile gematcht werden soll. Dabei kann man auch direkt festlegen, wie die Daten gespeichert werden sollen, falls gematcht wurde. Allgemein ist die Syntax dabei %{REGEXP:Feld}. Neben einigen regulären Ausdrücken, die im Grok Filter bereits implementiert sind (z.B. *LOGLEVEL, TIMESTAMP_ISO8601, SPACE, URIPATH*, uvm.), kann man auch selbst definierte reguläre Ausdrücke angeben (Hopf, 2016). Wie schon in Kapitel 2.2.1 beschrieben, sollen die Daten segmentiert werden, was mit dem Grok Filter realisiert werden kann. So kann man z.B. durch richtiges Platzieren von *Incoming Request* in dem regulären Ausdruck erreichen, dass *Outgoing Responses* nicht gematchet und dadurch auch nicht weiter beachtet werden. Außerdem ist es möglich Daten zu matchen, aber keinem Feld zuzuweisen, wodurch sie auch nicht gespeichert werden.
Weiterhin hat man die Möglichkeit die Felder, die durch das Matching mit dem regulären Ausdruck mit Daten gefüllt werden, weiter zu bearbeiten. Dafür können wiederum verschiedene Filter benutzt werden. So hat man die Option Felder, die Logstash automatisch anlegt, zu bearbeiten. Ein praktisches Beispiel hierfür ist der *date*-Filter (Hopf, 2016): Logstash legt automatisch ein Feld für einen Zeitstempel an, der auf den Moment eingestellt ist an dem Logstash den eingehenden Eintrag verarbeitet. Mit dem date-Filter hat man die Möglichkeit dieses Datum zu manipulieren. In unserem Fall wurde nicht der aktuelle Zeitstempel benutzt, um die Einträge zu speichern, sondern der Zeitstempel aus dem Logeintrag. Das hat den Vorteil, dass man nicht zwei verschiedene Zeitstempel speichern muss.

Ruby Filter-Plugin

Mit dem Ruby Filter-Plugin werden die transformierten Daten verarbeitet, um sie in ein passendes Format für Elasticsearch zu bringen. An dieser Stelle sei angemerkt, dass Elasticsearch zwar eine experimentelle *transform* Funktion besitzt, diese aber für unseren Anwendungsfall (noch) nicht passend implementiert ist (Elastic, 2020e). Deshalb wurde ein Ruby Skript entwickelt, das diese Funktion ergänzt.
Das entwickelte Skript liest aus JSON Dateien die Session Entities ein, die in Kapitel 2.2.2 vorgestellt wurden. Die Daten sind in drei Ebenen aufgeteilt:

1. UserIDs
2. SessionIDs

3. Widgets

Dabei kann man jeweils eine Ebene als ein Array betrachten. So besteht die oberste Ebene aus einem Array, das mit UserIDs gefüllt ist. Zu der UserID wird zusätzlich ein Array gespeichert, das aus SessionIDs besteht, die zu der UserID gehören. Zu diesen SessionIDs wird schließlich ebenfalls ein Array verknüpft, in dem festgehalten wird, welche Widgets in der Session genutzt wurden und wie oft. Nun durchläuft das Skript also die beschriebenen Array Ebenen und speichert die gegebenen Daten in passende Felder. Die Daten werden schließlich über Logstash an Elasticsearch gesendet und dort in einem passenden Index gespeichert (Elastic, 2020k).
Erwähnenswert ist noch, dass die transformierten Daten auf zwei Arten verarbeitet werden mussten. Die Gründe dafür werden in Kapitel 3.1.4 genauer erläutert.

3.1.3 Elasticsearch

Nachdem die von Filebeat gelesenen Daten von Logstash geparst wurden, werden die Daten in Elasticsearch gespeichert. Dabei werden die Logeinträge und die transformierten Daten separat indiziert. Die Daten, die an Elasticsearch gesendet werden, können dynamisch gemapped (also einem Datentyp zugeordnet) werden (Elastic, 2020c). Das heißt, Elasticsearch erkennt, ob z.B. ein Datum oder eine Zahl gespeichert wurde. Desweiteren hat man aber auch die Möglichkeit ein eigenes Mapping zu definieren (Elastic, 2020f). Nachdem aber ein Mapping in dem Index gespeichert wurde, kann es nicht mehr ohne weiteres geändert werden. Da laut Kapitel 3.1.2 in den Session Entities Arrays benutzen werden, muss darauf geachtet werden, dass diese in Elasticsearch vom Typ *nested* (Hopf, 2016) abgespeichert werden.
Für das weitere Verständnis ist es notwendig zu beschreiben, wie unsere Daten in Elasticsearch gespeichert werden. Um nicht unnötig ausschweifend zu werden, beschränken wir uns nur auf die Aspekte, die für uns relevant sind.
Allgemein werden Daten in Elasticsearch in Indizes gespeichert. Diese Indizes sind eine Sammlung von Dokumenten, in denen die gespeicherten Daten stehen. Die Daten sind als Key-Value-Paare gespeichert (Elastic, 2020b).

3.1.4 Kibana

Kibana ist ein Tool, um die Daten, die in Elasticsearch gespeichert sind, zu visualisieren bzw. zu analysieren. So kann man bspw. mit wenigen Einstellungen Visualisierungen der in Elasticsearch gespeicherten Daten erstellen, welche man wiederum zu einem Dashboard zusammenfassen kann. Ferner bietet Kibana die Möglichkeit die Daten auch manuell zu durchsuchen (Hopf, 2016). Sollte Kibana eine gewünschte Visualisierung oder ein gewünschtes Analysetool nicht nativ enthalten, hat man die Möglichkeit es um die gewünschten Funktionen zu erweitern (Elastic, 2020g).
Für unsere Anwendung ist eben dieser Fall eingetreten, da man nicht ohne weiteres Assoziationsregeln mit Kibana finden kann.
Bevor wir darauf eingehen, wie das Custom Plugin und die Visualisierungen eingesetzt werden, wollen wir den Hinweis aus Kapitel 3.1.2 aufgreifen und erläutern, warum es notwendig ist, die transformierten Daten in separaten Indizes zu speichern. Wie schon

erwähnt wird zu einer SessionID ein Array gespeichert das Informationen zu den benutzten Widgets enthält. In Elasticsearch kann man Arrays unter dem Datentyp *nested* speichern. Allerdings ist es in den Visualisierungen, die für unsere Zwecke relevant sind, nicht möglich Daten in einer verschachtelten Datenstruktur zu lesen bzw. zu durchsuchen (Elastic, 2020h). Also müssen die Einträge zu den transformierten Daten in einzelnen Dokumenten in Elasticsearch gespeichert werden.
Für die Suche nach Assoziationsregeln ist diese Datenstruktur allerdings nicht geeignet. Möchte man z.B. nach den Sessions suchen, in denen zwei bestimmte Widgets benutzt wurden, schlägt die Suche fehl. Die Begründung dafür ist, dass in den Dokumenten nur ein Widgetname steht. Deshalb müssen die Session Entities sowohl in der Array Variante gespeichert werden, als auch als Dokumente, die nur einen Widgetnamen beinhalten.

Visualisierungen

In dieser Arbeit wurden zwei Visualisierungen benutzt, die in Kibana bereits integriert sind: die *Pie* und *Line* Visualisierungen. Erstere wurde benutzt, um die anteilige Widgetnutzung darzustellen.
Die Line Visualisierung wurde benutzt, um die erste Frage aus Kapitel 1.2 zu beantworten. Die Visualisierung zeigt in einem zweidimensionalen Diagramm die Benutzung der Widgets über einen bestimmten Zeitraum an, wobei dieser Zeitraum auf der x-Achse abgebildet wird. Die y-Achse gibt die Anzahl an Widget Nutzungen an. Das bedeutet ein Punkt in dem Diagramm zeigt an, wie oft ein Widget an einem bestimmten Tag benutzt wurde. Schließlich verbindet die Visualisierung die Punkte miteinander und so erhält man eine Linie, die der Widgetnutzung in einem bestimmten Zeitraum entspricht. Abbildung 4 zeigt wie die Visualisierung beispielsweise aussehen könnte.

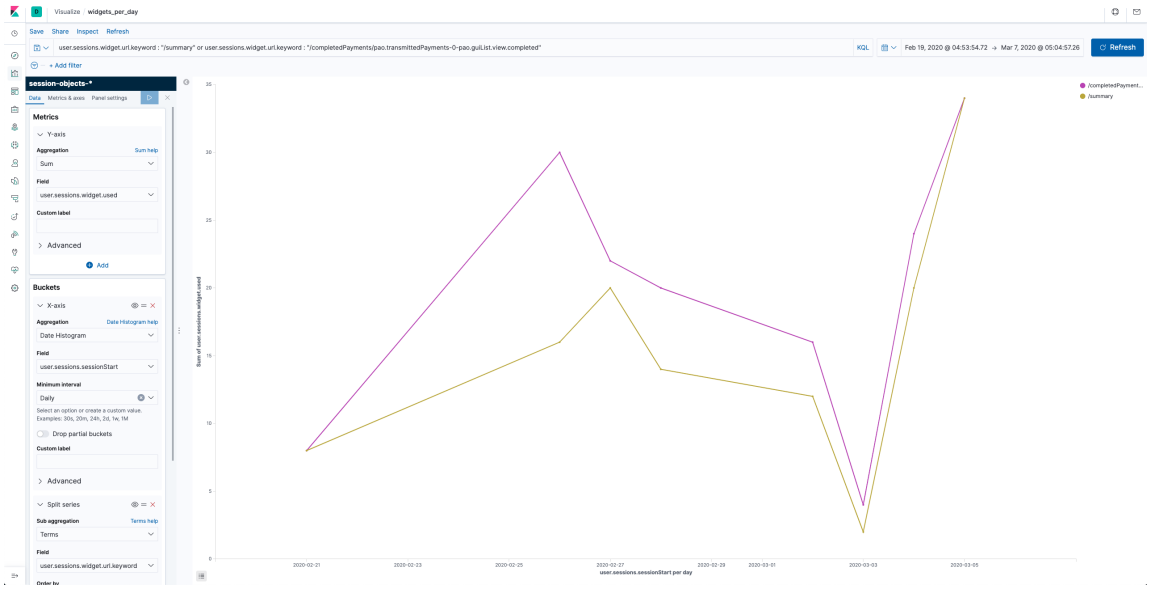

Abbildung 4: Lines Visualisierung mit zwei Widgets

Custom Plugin

Um einen erleichterten Einstieg in die Entwicklung eines Custom Plugins zu ermöglichen, bietet Kibana die Möglichkeit, das Grundgerüst für ein Custom Plugin generieren zu lassen (Elastic, 2020g). Das generierte Plugin liefert direkt die nötige Server-Client-Architektur, um den Elasticsearch-Server anzusprechen. Die grafische Oberfläche, die der Nutzer sieht, ist dabei als Client zu verstehen. Von hier aus werden HTTP Requests an einen Node.js Server gesendet. Dieser kann nun die gesendeten Daten (bspw. eine Suchanfrage) vorbereiten, an den Elasticsearch-Server senden, die Antwort verarbeiten und an den Client weiterleiten. Außerdem ist der Node.js Server auch in der Lage Python Code ausführen zu lassen. Die folgende Abbildung soll diese Beziehung verdeutlichen.

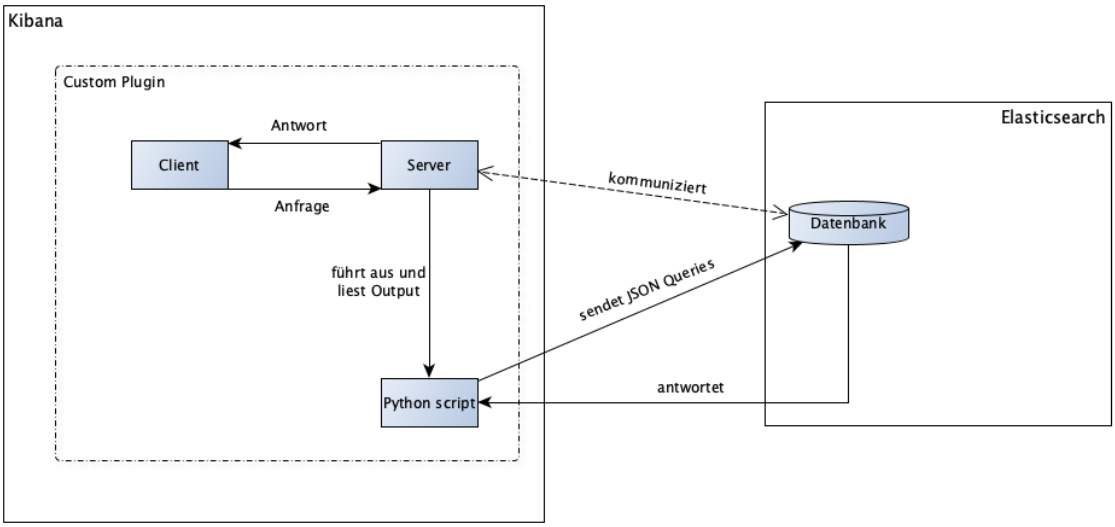

Abbildung 5: Architektur des Custom Plugins

Für den weiteren Verlauf der Arbeit sind die technischen Details der Client-Server-Architektur nicht weiter relevant. Von daher wird nun auf die Implementierung der Python Skripte zur Transformierung der Daten und zur Assoziationsregelanalyse eingegangen.

transform.py

Das Python Skript zur Transformierung der Daten wird durch das Frontend des Node.js Servers per Button ausgeführt. Bevor die Daten transformiert werden prüft das Skript, ob ein passender Index für die transformierten Daten in Elasticsearch existiert. Ist dies nicht der Fall, wird der entsprechende Index erstellt. Diesem Index wird bei der Erstellung das Mapping vorgegeben, was in Kapitel 3.1.3 erwähnt wurde. Da nun sicher gestellt ist, dass die transformierten Daten korrekt abgespeichert werden, holt sich das Skript eine Liste mit allen verfügbaren Indizes. Aus dieser Liste filtern wir die Indizes, die transformiert

werden sollen. An diese Indizes wird nun eine Searchquery gesendet, deren Antwort die transformierten Daten sind. Dabei wollen wir Duplikate und unvollständige Datensätze vermeiden. Ein Duplikat kommt zustande, wenn ein Index zwei Mal transformiert wird. Da Logstash bzw. Elasticsearch nicht prüft, ob Daten doppelt gespeichert werden, ist es möglich, dieselben Daten mehrmals in einem Index zu speichern. Deshalb prüft das Skript, ob für den Index bereits eine transformierte Datei existiert. Unvollständige Daten können auftreten, indem ein Index zu früh transformiert wird. Es könnte z.B. sein, dass man um 15 Uhr des aktuellen Tages die Daten transformiert, aber später noch neue Daten in den Index gespeichert werden. Würde man den Index zu einem späteren Zeitpunkt nochmal transformieren, hätte man wieder das Problem mit den Duplikaten. Die einfachste Lösung ist demnach, Transformierungen von tagesaktuellen Daten zu unterbinden.
Die transformierten Daten werden in demselben Ordner gespeichert, in dem die dazu gehörigen Logfiles liegen. Diese Information ist in dem zu transformierenden Index gespeichert. Dadurch wird sicher gestellt, dass Filebeat die transformierten Daten liest, da auch schon die Logfiles in diesem Ordner gelesen wurden.

associationRuleMiner.py

In diesem Abschnitt wird die Implementierung des Apriori Algorithmus aus Kapitel 2.2.3 beschrieben. Wie auch in Kapitel 2.2.3 basiert die Implementierung auf den Pseudocode aus „*Knowledge Discovery in Databases*“ von Ester und Sander (2000).
Zu Beginn des Skripts werden der minsupport und die minkonfidenz von der Kommandozeile bzw. von dem Node.js Server, der das Skript ausführt, eingelesen. Um eine Verbindung zu Elasticsearch aufzubauen, wird das Modul ELASTICSEARCH benutzt. Als nächstes wird über den Elasticsearch-Client die Größe der Datenbank[1] und eine Liste an benutzten Widgets ermittelt. Die Liste der Widgets wird als SET gespeichert und die Elemente als FROZENSET. Das hat den Vorteil, dass wir nun eine Kandidatenmenge haben, die aus einelementigen Mengen besteht. Aus dieser Menge werden nun mit dem Apriori Algorithmus die häufigen Itemsets ermittelt.
Erwähnenswert ist an dieser Stelle die Funktion *joinSets*. Anhand des folgenden Codeausschnitts wird die Funktionsweise erläutert.

[1] Datenbank wird in diesem Fall als Synonym für Anzahl an Sessions benutzt.

```
1  def joinSets(frequentItems):
2      """ Generates new candidate itemsets by joining the elements in frequentItems.
           It iterates through the itemset and
3          joins itemsets that have k-1 equal elements where k is the length of the
               itemsets. The result is a new
4          candidate set with k + 1 elements
5
6      Args:
7          frequentItems (set): itemset from which candidates are generated
8
9      Returns:
10           newCandidatesSet (set): Set of newly generated candidates
11
12     """
13     newCandidatesSet = set()
14
15     while True:
16         setIterator = iter(frequentItems)
17         try:
18             startItem = next(setIterator)
19             while True:
20                 try:
21                     nextItem = next(setIterator)
22                     if len(startItem.intersection(nextItem)) == len(startItem) - 1:
23                         newCandidatesSet.add(startItem.union(nextItem))
24                 except StopIteration:
25                     frequentItems.remove(startItem)
26                     break
27         except StopIteration:
28             break
29
30     return newCandidatesSet
```

Listing 3: joinSets aus associationRuleMiner.py

In Zeile 15 - 28 wird mit Hilfe eines Iterators durch die Menge iteriert, deren Elemente die Länge k haben. Das erste Element der Menge wird dabei in der Variable *startItem* gespeichert. Nun wird für jedes weitere Element der Menge geprüft, ob die beiden Mengen miteinander vereinigt werden können. Die Voraussetzung dafür ist, dass beide Mengen $k-1$ gleiche Elemente haben. Diese Voraussetzung wird in Zeile 22 geprüft. Wenn der Durchschnitt der beiden Mengen die Länge $k-1$ hat, bedeutet das, dass die beiden Mengen genau $k-1$ Elemente haben. Also dürfen diese Mengen vereinigt und in *newCandidatesSet* gespeichert werden. Wenn alle Elemente durchlaufen wurden, wird das *startElement* aus der Menge über die iteriert wird gelöscht. Dadurch steht das ursprünglich zweite Element an erster Stelle. Der Iterator wird entsprechend aktualisiert und es wird versucht, dieses Element mit einem anderen zu vereinigen. Hier wird nun deutlich, worin der Vorteil bei der Wahl von SET und FROZENSET als Datentyp liegt: durch die Mengenoperationen wurde vermieden, dass die Mengen elementweise verglichen werden müssen.[2]
Anschließend muss für die vereinigten Mengen geprüft werden, ob alle ihrer Teilmengen häufig sind. Dazu berechnet die Funktion IT.COMBINATIONS aus dem Modul ITERTOOLS alle $k-1$ elementigen Teilmengen. Für jede Teilmenge wird geprüft, ob sie in der Menge der häufigen Itemsets enthalten ist. Ist dies nicht der Fall wird das Element, das die

[2]Python implementiert SETs als Hash Tables. Intern werden z.B. bei der Vereinigung von Mengen auch Elemente miteinander verglichen (Geeks for Geeks, 2020). Aber da es sich um Hash Tables handelt geht dies schneller, als würde man das selber implementieren.

Teilmenge beinhaltet gelöscht. Daraus resultiert die neue Kandidatenmenge, für die der Support geprüft wird.
Nachdem die häufigen Itemsets bestimmt wurden, werden alle Regeln gesucht, die die minkonfidenz erfüllen. Die Implementierung dazu zeigt Listing 4.

```
1  def generateRules(freqItems, antecedenceItems, resultSet):
2      """ This function generates all possible association rules that can be found in
           the frequent itemsets that satisfy
3          the minconf. Like the minsupport the minconf need to be passed by the
               command line using -minconf=[value].
4          The confidence of a rule A => X - A is calculated as support(X) / support(A)
               . If this rule satisfies the
5          minconf it will check if A' also satisfies the minconf where A' is a subset
               of A. This is done recursively
6          at the end of the function.
7
8      Args:
9          freqItems (set): Set of frequent items in which association rules should be
               found
10         antecedenceItems (set): Set of items that are a a subset of the itemset of
               the left side of an association rule
11         resultSet (set): Set of found association rules
12
13     """
14     for item in antecedenceItems:
15         item = antecedenceItems.difference(frozenset([item]))
16
17         supportItemset = getCount(freqItems) / dbsize
18         supportItem = getCount(item) / dbsize
19
20         confidence = supportItemset / supportItem
21
22         if confidence >= float(args.minconf):
23             consequence = freqItems.difference(item)
24             resultStr = f'{list(item)}->{list(consequence)},' \
25                         f'support:{format(supportItemset,".4f")},confidence:{format
                               (confidence,".4f")}'
26             resultSet.add(resultStr)
27             if len(item) > 1:
28                 generateRules(freqItems, item, resultSet)
29
30 for frequentItem in frequentItems:
31         generateRules(frequentItem, frequentItem, resultSet)
```

Listing 4: generateRules aus associationRuleMiner.py

Initial wird die Funktion GENERATERULES mit einem häufigen Itemset aufgerufen, wobei dieses Itemset zwei Mal als Argument übergeben wird. Da wir Regeln der Form

$$Y \rightarrow X - Y, Y \subseteq X$$

bilden wollen, wird die Menge Y (Zeile 1: *antecedenceItems*) elementweise durchlaufen und die Differenz zur Menge X (Zeile 1: *freqItems*) gebildet (Zeile 15). Die Konfidenz solch einer Regel kann nach Ester und Sander (2000) durch

$$\text{konfidenz}(Y \rightarrow (X - Y)) = \frac{\text{support(X)}}{\text{support(Y)}}$$

berechnet werden (Zeile 20). Falls diese Regel die Konfidenz erfüllt, wird dies in RESULTITEM gespeichert. Anschließend werden rekursiv Regeln für Y', $Y' \subset Y$ gesucht.

4 Auswertung

In diesem Kapitel wird beschrieben, wie das entwickelte System eingesetzt und getestet wurde. Zuerst wird dargelegt, mit welchen Daten das System getestet wird. Als nächstes wird der Ablauf zur Durchführung des Tests beschrieben. Eine kritische Beleuchtung der Ergebnisse schließt das Kapitel ab.

4.1 Datengenerierung

Die zur Analyse genutzten Daten sind keine tatsächlichen Kundendaten, sondern wurden zum Zweck dieser Abschlussarbeit generiert. Das war notwendig, da in der aktuellen Version des IFPs die Widgetnamen in der URL nicht festgehalten werden. Dadurch ist es nicht möglich, die Widgets anhand der URL eindeutig zu bestimmen. Selbst bei einer schnellen Auslieferung wäre der Zeitraum zu klein, um genügend Kundendaten zu sammeln.
Mithilfe eines Python Skripts wurden Logfiles für einen Zeitraum von 90 Tagen generiert. Die generierten Logfiles entsprechen einer verkürzten Version der normalen Logfiles: es wurden nur *Incoming Requests* und Trennlinien, wie sie auch in den echten Logfiles vorkommen, in die generierten Dateien geschrieben. Zu Beginn der Entwicklung des Systems wurde der Logstash Filter mit Logfiles entworfen, in denen alle Einträge standen, also auch *Outgoing Responses*. Zu diesem Zeitpunkt hat der Filter erfolgreich die Daten nach unseren Wünschen aus Kapitel 2.2.1 segmentiert. Deshalb war es für die Generierung der Daten ausreichend nur noch *Incoming Requests* zu betrachten, die eine Widgetnutzung repräsentieren. Insgesamt wurden 15 verschiedene Widgets betrachtet.
Um zu prüfen, ob das entwickelte System nach unseren Wünschen funktioniert, müssen die Logfiles zwei Bedingungen erfüllen:

1. Die Einträge in den Logfiles müssen zufällig ausgewählt werden. Die Idee hinter dieser Bedingung ist, dass die Daten in den Logfiles unbekannt sind.

2. Eine Kombination an Widgets muss in verschiedenen Abständen wiederholt in den Logfiles vorkommen. Diese Kombination stellt einen bestimmten Workflow dar, der durch die Suche nach Assoziationsregeln erkannt werden soll. Die einzige Information, die wir zu diesem Punkt verwenden möchten, ist die Anzahl an Sessions und die Anzahl an Sessions, die diesen Workflow beinhalten. Durch diese Werte lässt sich der minsupport für den Workflow abschätzen und dient als Hilfe zur Erkennung des Workflows.

Unter Berücksichtigung dieser Bedingungen wurde das Skript entwickelt. Eine Ausführung des Python Skripts entspricht einer Session. In dieser Session werden die Widgets, die „benutzt“ werden, zufällig ausgewäht. Das wurde erreicht, indem die beispielhaften *Incoming Requests* als Strings in einem Array gespeichert werden. Pythons RANDOM Funktion liefert eine zufällige Zahl in den Array Grenzen, die bestimmt welche Zeile geschrieben wird. Zusätzlich hat man die Möglichkeit bei der Ausführung des Skripts

eine Reihe an Zahlen zu übergeben, die den Workflow darstellen. Schließlich kann über einen weiteren Parameter eine Zahl übergeben werden, die angibt welches Datum geloggt wird. Dieser Wert ist standardmäßig auf 0 gesetzt, was dem aktuellen Datum entspricht. Falls dieser Parameter übergeben wird, wird die Zahl zum aktuellen Datum addiert.
Zusätzlich zu dem Python Skript wurde ein Bash Skript entwickelt, welches die gesamte Datengenerierung automatisiert. Das Skript simuliert für einen festgelegten Zeitraum (in unserem Fall 30 Tage) eine zufällige Anzahl an Sessions pro Tag. Außerdem soll ca. jede dritte Session den festen Workflow beinhalten. Welche Widgets benutzt werden, wird durch drei zufällige Zahlen bestimmt. Am Ende gibt das Skript auf der Kommandozeile aus, wie viele Sessions insgesamt geschrieben wurden und wie viele davon den Workflow beinhalten.

4.1.1 Durchführung

Als erstes wurde begonnen die Logfiles zu generieren, während dieses Vorgang wurde auch der ELK gestartet. Die Vorgänge laufen parallel ab, um den echten Anwendungsfall zu simulieren. Der echte Anwendungsfall ist, dass Daten geloggt werden während das IFP läuft, es aber auch schon Daten aus den vergangenen Tagen gibt. Nach der Generierung der Daten wird das Custom Plugin verwendet, um die Daten mithilfe des Pythonskripts zu transformieren.
Die transformierten Daten werden von Filebeat gelesen, von Logstash geparst und in den entsprechenden Indizes in Elasticsearch gespeichert. Damit sind die Daten für die Visualisierungen und die Suche nach Assoziationsregeln vorbereitet.
Um die Daten visualisieren zu können, muss man vorher in Kibana ein Index Pattern anlegen (Elastic, 2020i). Damit wird die Visualisierung aller Indizes berücksichtigt, die dem Pattern entsprechen. Um zu testen, ob die Transformierung erfolgreich war, werden zwei Pie Diagramme miteinander verglichen. Für die untransformierten Daten hat die Pie Visualisierung gezählt, wie oft ein Widget pro Index vorkommt und anteilig dargestellt. In den transformierten Daten haben wir die Information, wie oft ein Widget in einer Session benutzt wurde. Wenn man diesen Wert über alle Session Entities für ein Widget summiert und das gleiche Diagramm entsteht, zeigt das, dass die Daten korrekt transformiert wurden. In der Lines Visualisierung wird dann entsprechend die Summe der Nutzungen pro Tag aufgeführt.
Die Suche nach den Assoziationsregeln wird in dem Custom Plugin durchgeführt. Dazu wird der minsupport so gewählt, dass der festgelegte Workflow gefunden werden müsste. Der tatsächliche minsupport des Workflows kann auch höher sein als der vom Skript zu Datengenerierung errechnete, da auch per Zufall der Workflow in einer Session vorkommen kann. Es ist zu erwarten, dass wir Regeln erhalten, die eine Permutation des Workflows darstellen. Die Wahl der minkonfidenz entscheidet dabei, wie tief diese Permutation geht. Das bedeutet, dass man bei einem Workflow mit drei Widgets mindestens drei Regeln findet. In dem Custom Plugin kann man den minsupport und die minkonfidenz über ein Nummernfeld festlegen und per Buttonclick die Suche starten.
Zusätzlich zu den Ergebnissen der Suche wird eine Analyse des Speicherverbrauchs des Skripts durchgeführt, mit Fokus auf die Funktionen zur Bestimmung der häufigen Itemsets und der Generierung der Regeln. Dazu wird das Python Modul MEMORY_PROFILER

(Pedregosa und Gervais, 2020) benutzt. Hierbei werden vier Szenarien analysiert, die sich in der Festlegung des minsupports und der minkonfidenz unterscheiden.

4.1.2 Resultate

An dieser Stelle werden die Resultate nach der Durchführung aus Kapitel 4.1.1 dargestellt. Dazu werden zunächst die Pie Diagramme (Abbildung 8 und Abbildung 9) in Anhang Teil B vorgestellt. Wie man erkennen kann, sind die beiden Diagramme identisch. Das bedeutet, dass die Lines Visualisierung aus Kapitel 3.1.4 den Verlauf der Widgetnutzung korrekt darstellt.
Wenden wir uns nun den Assoziationsregeln zu. Das Skript zur Generierung der Daten hat insgesamt 914 Sessions geschrieben, wovon 275 Sessions einen festen Workflow enthalten. Somit ergibt sich ein minsupport von mindestens 0.30 [3]. Als minkonfidenz wurde 0.80 gewählt. Abbildung 6 zeigt welche Regeln gefunden wurden:

Abbildung 6: Ergebnis der Suche nach Assoziationsregeln

Wie sich erkennen lässt, tritt das Itemset *IncomingTransactionWidgetContent, BalancesWidgetContent* und *LiquidityByAccountsWidgetContent* bzgl. des angegebenen minsupports häufig auf. Desweiteren stellen die Regeln eine Permutation dar, in der jedes Widget ein Mal als Konsequenz vorkommt. Daraus lässt sich ableiten, dass die Implementierung zum Finden der Assoziationsregeln korrekt funktioniert.
Aus der Analyse des Speicherverbrauchs resultieren die folgenden Plots:

[3] Der tatsächliche minsupport der Regel kann auch höher sein, da das Itemset auch zufällig in einer Session vorkommen kann

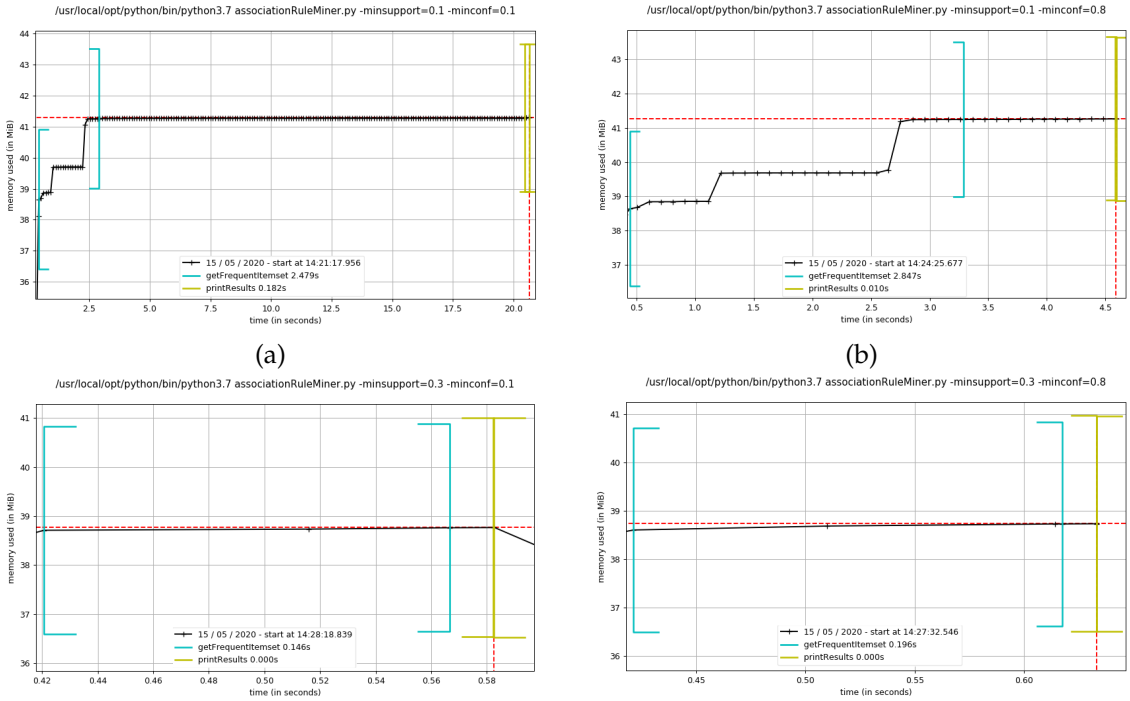

Abbildung 7: Ergebnis Speicheranalyse

Die Plots zeigen den Speicherverbrauch ab dem Zeitpunkt, an dem die häufigen Itemsets bestimmt werden (blaue Klammer) bis zur Ausgabe der gefundenen Regeln (gelbe Klammer)[4] an. Der Bereich zwischen den Klammern stellt den Speicherverbrauch während der Generierung der Regeln dar. Die Parameter für die Werte minsupport und minkonfidenz wurden so gewählt, dass sie einmal einen eher schlecht gewählten Wert bekommen (0.1) und entsprechend passendere Werte (minsupport = 0.3, minkonfidenz = 0.8). Es ist trivial, dass sich der gewählte minsupport auf die blaue Klammer auswirkt und die minkonfidenz auf die gelbe Klammer. Allerdings wirkt sich ein unpassend gewählter minsupport vergleichsweise stark auf die Speichernutzung aus. Vergleichen wir dazu einmal Abbildung 7b und Abbildung 7d. Wie sich deutlich erkennen lässt, wird bei einem niedrigen minsupport wesentlich mehr Speicher benötigt, da mehr Kandidaten für häufige Itemsets betrachtet werden müssen. Dementsprechend dauert es auch länger alle häufigen Itemsets zu bestimmen. Betrachtet man den Bereich zwischen der blauen und gelben Klammer, der die Generierung der Regeln darstellt, lässt sich festestellen, dass sich die Wahl der minkonfidenz mehr auf die benötigte Zeit auswirkt, um Regeln zu finden, als auf die Speichernutzung. Das sieht man besonders, wenn man Abbildung 7a und Abbildung 7b miteinander vergleicht: im ersten Fall müssen wesentlich mehr Funktionsaufrufe getätigt werden, erkennbar an den schwarzen, senkrechten Strichen auf der schwarzen Linie.

[4] Für die Generierung der Assoziationsregeln wurde nicht direkt die Speichernutzung gemessen, sondern durch die Funktionen getFrequentItemsets und printResults eingegrenzt. Diese Funktionen werden unmittelbar vor bzw. nach der Generierung der Regeln aufgerufen. Der Grund für diese Vorgehensweise ist, dass MEMORY_PROFILER pro Funktionsaufruf eine Klammer darstellt. Da die Regeln rekursiv gefunden werden, wären zu viele Klammern in dem Plot erschienen, was die Lesbarkeit stark einschränkt.

4.1.3 Diskussion

In diesem Abschnitt sollen die Ergebnisse aus Kapitel 4.1.2 kritisch diskutiert werden.
Wenden wir uns dazu zunächst den Visualisierungen zu. Anscheinend ist es für die Visualisierung nicht zwingend notwendig, die Daten zu transformieren. Das folgt aus der Tatsache, dass es Möglich war, das selbe Diegramm aus zwei verschiedenen Indices zu erstellen.
Dennoch hat die Transformierung den Vorteil, Daten in einem Format zu zeigen, welches sich für weitere Analyseverfahren in Bereich *Knowledge Discovery in Databases* (KDD) eignet. Denkbar wäre hier die Anwendung der Nächste-Nachbarn-Klassifikation (Ester und Sander, 2000) auf die Arrays, welche die Widgetnutzung pro Session speichern.
Im Bezug auf die Suche nach den Assoziationsregeln war es ebenfalls von Vorteil, die Daten transformiert zu betrachten. Durch die transformation hat ein besseres Verständnis über Daten erhalten.
Bevor die Ergebnisse der Assoziationsregelnanalyse diskutiert werden, ist es wichtig zu klären, welche Information eine Assoziationsregel genau liefert. Nach Beierle und Kern-Isberner (2019) kann man den minsupport als relative Häufigkeit und die minkonfidenz als die Wahrscheinlichkeit einer Regel betrachten. Vergleichen wir dazu die erste Regel aus Abbildung 6. Diese Regel sagt aus, dass in 39.62% aller Sessions die Widgets *IncomingTransactionWidgetContent, BalancesWidgetContent* und *LiquidityByAccountsWidgetContent* zusammen vorkommen. Mit einer Wahrscheinlichkeit von 84.52% wird *LiquidityByAccountsWidgetContent* angeklickt, wenn die anderen beiden Widgets benutzt wurden. Die Assoziationsregel suggeriert eine temporale Beziehung zwischen Antezedenz und Konsequenz der Regel. Tatsächlich berücksichtigt der in dieser Arbeit verwendete Algorithmus keine zeitlichen Aspekte, sodass dieser Rückschluss falsch sein kann (Beekmann, 2003).
Nichtsdestotrotz kann man bei realen Kundendaten vermuten, dass auch eine zeitliche Beziehung zwischen den Widgets besteht. Wird bspw. die Regel

$$\textit{Open Payments} \rightarrow \textit{LiquidityByAccountsWidgetContent}$$

gefunden, die den minsupport und die minkonfidenz erfüllt, ist es naheliegend, dass *Open Payments* zuerst benutzt wurde. Diese Interpretation basiert aber ausschließlich auf der Vermutung des Benutzers.
Aus dieser Feststellung folgt eine weitere Eigenschaft der Assoziationsregeln, die erwähnt werden muss. Da zeitliche Abfolgen nicht berücksichtigt werden, kann es auch durchaus sein, dass ein Workflow „unterbrochen“ wird. Das bedeutet, dass zwischen der Antezedenz und Konsequenz einer Regel auch weitere Widgets benutzt worden sein können. Diese Tatsache wirkt zunächst wie eine Einschränkung für die Verwendbarkeit bzw. Aussagekraft der Assoziationsregeln. Man kann dies aber auch als Vorteil betrachten, da der Algorithmus Workflows trotz Unterbrechungen findet. So könnte eine Unterbrechung z.B. aus Versehen zustande kommen, indem sich ein User „verklickt“ hat.
Mit Blick auf die Aufgabenstellung aus Kapitel 1.2 lässt sich also zusammenfassen, dass die Suche nach Assoziationsregeln Workflows erkennen kann, mit der Voraussetzung, dass zeitliche Aspekte nicht berücksichtigt werden.

5 Zusammenfassung

In dem letzten Kapitel der Arbeit wird ein Fazit mit Blick auf die Ergebnisse aus dem vorherigen Kapitel gezogen. Schließlich werden zukünftige Anwendungsmöglichkeiten aufgezeigt, die sich auf drei Bereiche beziehen: Umwandlung, Optimierung und Erweiterung. In dem Abschnitt zur Umwandlung werden Denkansätze vorgestellt, wie man das System umwandeln kann, um es in neuen Bereichen einzusetzen. Anschließend geht es in dem Abschnitt zur Optimierung darum Stellen aufzuzeigen, welche ein Verbesserungspotential des entwickelten Systems bergen. Zum Schluss wird eine Aussicht auf Funktionalitäten gegeben, um die das System erweitert werden kann.

5.1 Fazit

Die Ergebnisse aus Kapitel 4 haben gezeigt, dass sich die im Rahmen dieser Arbeit vorgestellten Methoden durchaus eignen, Logfiles bzgl. des Userverhaltens zu analysieren. Trotzdem ist es wichtig an dieser Stelle anzumerken, dass die aktuelle Version des IFPs noch nicht die notwendigen Informationen in den Logfiles enthält und somit das System nicht direkt zur Analyse eingesetzt werden kann. Erst wenn in den Logfiles des IFPs bzw. in den URLs der Incoming Requests der Widgetname eindeutig festgehalten wird, ist dieses System einsetzbar. Es ist zwar durchaus möglich, einige Widgets anhand der Parameter in den URLs zu erkennen, aber damit würde man nur eine Teilmenge der Widgets abdecken, die zum Einsatz kommen.
Ebenso sei an dieser Stelle erwähnt, dass die Mittel und Wege zur Datenanalyse in dieser Arbeit nur eine mögliche Variante sind, an das erwünschte Ziel zu kommen. Während der Umsetzung der technischen Aspekte der Arbeit, wurden mehrere Ansätze ausprobiert die gegebenen Probleme zu lösen. Als Beispiel lässt sich das Ruby Filter-Plugin aus Kapitel 3.1.2 nennen. Anstelle des hier angewendeten Plugins kan man auch das JSON Filter-Plugin benutzen, um die transformierten Daten zu parsen. Da aber mit dem Ruby Filter-Plugin das gewünschte Ergebnis durch Ausprobieren schneller erreicht wurde, wurde dieses letzlich auch verwendet.

5.2 Zukünftige Anwendungsmöglichkeiten

Um diese Arbeit abzuschließen werden zukünftige Anwendungsmöglichkeiten vorgestellt. Zunächst werden Möglichkeiten vorgeschlagen, wie man die im Rahmen der Arbeit entwickelten Analysemöglichkeiten auf weitere Aspekte des IFPs umwandeln kann. Daran anknüpfend werden mögliche Optimierungen des System angeboten. Schließlich wird erläutert, wie man das System mit neuen Funktionalitäten erweitern könnte.

5.2.1 Umwandlungen

Wie bereits in Kapitel 4.1.3 erwähnt, bietet sich durch die eingeführte Datenstruktur der Session Entities die Möglichkeit, vektorbasierte Analysen durchzuführen. Neben der Nächste-Nachbarn-Methode wäre es z.B. auch denkbar Widgets basierend auf der

Häufigkeit der Nutzung zu klassifizieren. Eine solche Klassifikation kann man wiederum nutzen, um zu bestimmen, welche Widgets nicht so oft benutzt werden und damit auch nicht so beliebt sind. Basierend auf dieser Information besteht die Möglichkeit das Widget zu überarbeiten.

Bis jetzt wurde vorgestellt, wie man die Logfiles des IFPs nutzen kann, um das Kundenverhalten bzgl. der Widgetnutzung zu analysieren. Da das IFP aber einen größeren Funktionsumfang hat, ist es naheliegend die Analyse auf weitere Bereiche zu erweitern. Eine Möglichkeit wäre einen Workflow nicht nur auf Widgets zu beschränken, sondern alle Seiten, die in einer Session aufgerufen wurden, zu betrachten.

In Kapitel 4.1.3 wurde bereits erörtert, dass die Assoziationsregelngenerierung keine zeitlichen Abfolgen berücksichtigt. Dies könnte man durch eine Modifikation des Algorithmus ändern. (Beekmann, 2003)
Desweiteren wäre es auch denkbar, die Widgets in hierarchische Strukturen einzuteilen und diese bei der Analyse zu berücksichtigen. Man könnte den Algorithmus auch dahingehend modifizieren, dass die Quantität der Widgets berücksichtigt wird (Ester und Sander, 2000)

5.2.2 Optimierung

An dieser Stelle sei drauf hingewiesen, dass es zumindest einen Aspekt der Arbeit gibt, der in Zukunft optimiert werden sollte. In Kapitel 3.1.4 wurde erläutert, dass für die transformierten Daten zwei Indizes nötig seien, um die Visualisierungen darzustellen und die Datenstruktur der Session Entities akkurat in Elasticsearch zu speichern. Aus den Ergebnissen in Kapitel 4.1.3 ist aber deutlich geworden, dass der extra Index für die Visualisierung der transformierten Daten redundant ist. Aus diesem Grund sollte in Zukunft darauf verzichtet werden, diesen Index anzulegen. Eine weitere Möglichkeit besteht darin, die transformierten Daten weder als JSON Dateien, noch in einem Index zu speichern. Stattdessen können die transformierten Daten temporär gecached werden. Es ist an dieser Stelle allerdings schwierig abzuschätzen, ob und wie vorteilhaft das wäre.

5.2.3 Erweiterungen

Abschließend wollen wir anregen, wie man das System um Funktionen erweitern kann, die von Elasticsearch angeboten werden. Da diese Möglichkeiten die kostenpflichtige Version von Elasticsearch voraussetzen, wurde im Rahmen dieser Arbeit zunächst darauf verzichtet, diese zu benutzen. So ist es sicherlich lohnenswert, sich Elasticsearchs Machine Learning Modul anzuschauen. Dort könnten ähnliche Analysemöglichkeiten enthalten sein, wie die im Kapitel 5.2.1 vorgestellten (Elastic, 2020d).
Außerdem könnte die Alert Schnittstelle in Elasticsearch von Interesse sein. Mit dieser Erweiterung hat man die Möglichkeit Benachrichtigungen zu versenden, falls ein vom User definiertes Ereignis eintritt (Elastic, 2020a).

A Anhang

Beispielrechnung aus Kapitel 2.2.3 mit minsupport = 0.3

w_1	0.6
w_2	0.8
w_3	0.4
w_4	0.2
w_5	0.6

prüfe minsupport →

w_1	0.6
w_2	0.8
w_3	0.4
w_4	0.2

join Itemsets (prune ergibt keine Änderung) →

w_1, w_2	0.4
w_1, w_3	0.2
w_1, w_5	0.2
w_2, w_3	0.2
w_2, w_5	0.6
w_3, w_5	0.2

prüfe minsupport →

w_1, w_2	0.4
w_2, w_5	0.6

join Itemsets → $\{w_1, w_2, w_5\}$

Da w_1, w_5 nicht häufig ist, ist auch $\{w_1, w_2, w_5\}$ nicht häufig. Somit sind w_1, w_2 und w_2, w_5 die gefundenen häufigen Itemsets..

B Pie Diagramme

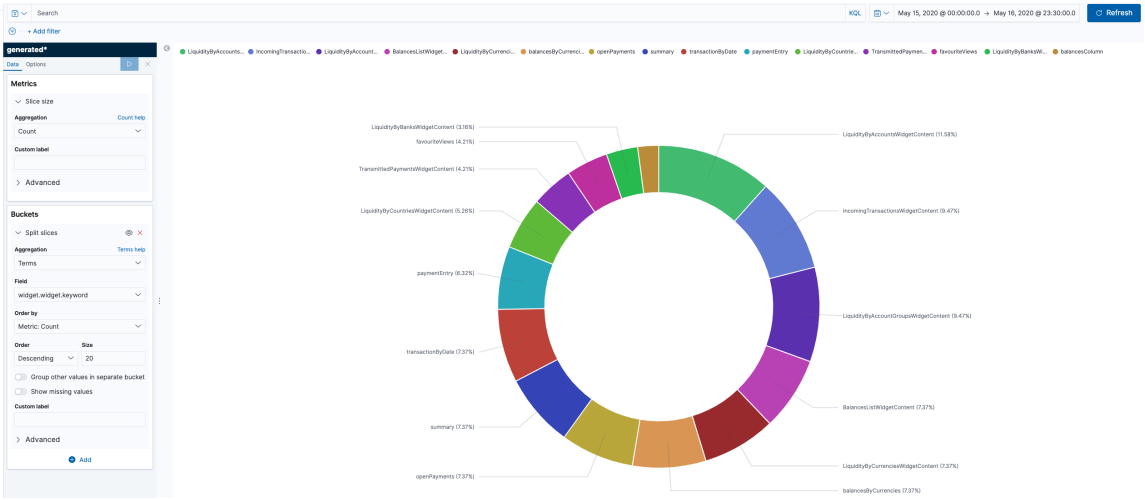

Abbildung 8: Pie Visualisierung der generierten Daten

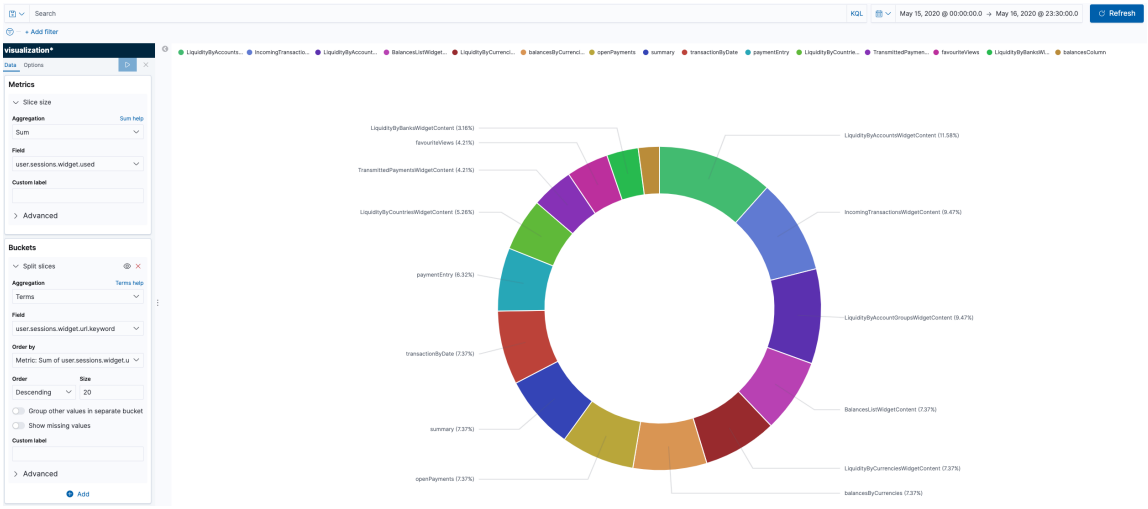

Abbildung 9: Pie Visualisierung der transformierten Daten

Literatur

Rakesh Agrawal, Tomasz Imielinski und Arun Swami (Juni 1993). „Mining Association Rules between Sets of Items in Large Databases“. In: *SIGMOD Rec.* 22.2, S. 207–216.

Frank Beekmann (2003). *Stichprobenbasierte Assoziationsanalyse im Rahmen des Knowledge Discovery in Databases*. Deutscher Universitätsverlag.

Christoph Beierle und Gabriele Kern-Isberner (2019). „Maschinelles Lernen“. In: *Methoden wissensbasierter Systeme: Grundlagen, Algorithmen, Anwendungen*. Springer Fachmedien Wiesbaden, S. 99–160.

CoCoNet (2020). *(CoCoNet)*. URL: https://www.coconet.de/en/corporate-banking-solutions/payment-cash-management-solution/ (besucht am 05. 05. 2020).

Richard O. Duda, Peter E. Hart und David G. Stork (2001). *Pattern Classification*. John Wiley & Sons, Inc.

Elastic (2020a). *Elasticsearch Alert*. URL: https://www.elastic.co/de/what-is/elasticsearch-alerting (besucht am 03. 05. 2020).

Elastic (2020b). *Elasticsearch Document Indices*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/documents-indices.html (besucht am 26. 02. 2020).

Elastic (2020c). *Elasticsearch Dynamic Mapping*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/dynamic-mapping.html (besucht am 14. 04. 2020).

Elastic (2020d). *Elasticsearch Machine Learning*. URL: https://www.elastic.co/de/what-is/elasticsearch-machine-learning (besucht am 03. 05. 2020).

Elastic (2020e). *Elasticsearch Transform*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/transforms.html (besucht am 07. 04. 2020).

Elastic (2020f). *Elastucsearch Explicit Mapping*. URL: https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html (besucht am 14. 04. 2020).

Elastic (2020g). *Kibana Custom Plugin*. URL: https://www.elastic.co/guide/en/kibana/current/development-plugin-resources.html (besucht am 31. 03. 2020).

Elastic (2020h). *Kibana Forum*. URL: https://discuss.elastic.co/t/getting-around-kibanas-missing-support-for-nested-objects-parent-child/53529 (besucht am 03. 04. 2020).

Elastic (2020i). *Kibana Index Pattern*. URL: https://www.elastic.co/guide/en/kibana/current/index-patterns.html (besucht am 26. 02. 2020).

Elastic (2020j). *Logstash filter plugins*. URL: https://www.elastic.co/guide/en/logstash/current/filter-plugins.html (besucht am 18. 02. 2020).

Elastic (2020k). *Ruby Filter Plugin*. URL: https://www.elastic.co/guide/en/logstash/current/plugins-filters-ruby.html (besucht am 04. 03. 2020).

Martin Ester und Jörg Sander (2000). *Knowledge Discovery in Databases*. Springer-Verlag, Berlin, Heidelberg.

Geeks for Geeks (2020). *Internal working of Set in Python*. URL: https://www.geeksforgeeks.org/internal-working-of-set-in-python/ (besucht am 14. 05. 2020).

Florian Hopf (2016). *Elasticsearch*. dpunkt.verlag GmbH.

Fabian Pedregosa und Philippe Gervais (2020). *Memory Profiler*. URL: https://github.com/pythonprofilers/memory_profiler (besucht am 15. 05. 2020).

Pranav Shukla und Kumar M. N. Sharath (2017). *Learning Elastic Stack 6.0 : a beginner's guide to distributed search, analytics, and visualization using Elasticsearch, Logstash, and Kibana*. Packt Publishing Ltd.

Patrick S.P. Wang (2011). *Pattern Recognition, Machine Intelligence and Biometrics*. Higher Education Press, Beijing und Springer-Verlag Berlin, Heidelberg.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings