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. Stefan Conrad

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

Hier kommt eine ca. einseitige Zusammenfassung der Arbeit rein.

Inhaltsverzeichnis

1 Einleitung

1.1 Firmenprofil CoCoNet GmbH

1.2 Problemstellung (Aufgabestellung?) (anforderungen an das system)

In dieser Arbeit geht es darum, das Kundenverhalten in Hinblick auf die Nutzung der Dashboard Widgets im IFP zu analysieren. Da der Begriff Kundenverhalten noch recht breit gefächert ist sollen konkret die folgen 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 zu 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 pure Vorhandensein auf dem Dashboard eine Nutzung schon einschließt. Betrachtet man z.B. das Widget in [ABB-SCREENSHOT-WIDGET] 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. Da also ein pures Vorhandensein des Widgets auf dem Dashboard als Nutzung nicht hinreichend ist, zählt erst ein Klick auf das Widget als Nutzung.
Wenn nun ein Widget angeklickt wird, wird der Nutzer in der Regel auf eine neue Seite weiter geleitet, 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 diese Information mit Blick auf die erwähnten Fragen entsprechend verarbeitet wird.

1.3 Aufbau der Arbeit

Im folgenden Verlauf der Arbeit wird zunächst auf die theoretischen Grundladen der Mustererkennung eingegangen. Dazu wird zunächst die Syntax der vorliegenden Logfile Einträge und anschließend die Phasen der Mustererkennung, die die Einträge durchlaufen, beschrieben. In dem darauf folgenden Kapitel wird beschrieben, wie die theoretischen Grundlagen technisch umgesetzt werden. Danach werden die Ergebnisse des entwickelten System ausgewertet. Zum Schluss wird die Arbeit durch ein Fazit zusammen gefasst und zukünftige Einsatzmöglichkeiten erörtert.

2 Grundlagen

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“ relavant sind, soll auch nur deren Syntax beschrieben werden.
Die Einträge in den Sessionlogs spiegeln im Großen und Ganzen die Serveraktivität wieder: Es werden ein- und ausgehende HTTP Requests fest gehalten. Allerdings reicht es nicht 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, wer wann welche Daten gesendet bzw. angefragt hat. Neben der UserID wird auch die SessionID 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 der selbe User nach dem Logout direkt noch einmal einloggen wird eine neue SessionID generiert. Die folgende Abbildung zeigt die die Requests, die beim Login Prozess geloggt werden: (ABB:LOGin-SESSIONLOG)
Man erkennt schnell, dass man die Einträge in den Logfiles grob in die folgenden Felder unterteilen kann:

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

Tabelle 1: Felder in Sessionlog Eintrag

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

2.2 Mustererkennung

Generell kann man in vielen verschiedenen Medien Muster erkennen, wie z.B. Bilddateien, Musikdateien oder eben in Textdateien. Dabei kommt es im Wesentlichen drauf an, welche Muster man erkennen möchte. So könnte man einem Programm beibringen, bestimmte Fischarten auf Fotos zu erkennen (Duda et al., 2001). Ein Ansatz, das Problem der Mustererkennung zu lösen, ist es, das Problem in folgende Teilprobleme aufzuteilen (Zitat anderes buch):

1. Segmentiereung
2. Feature Extraction
3. Klassifizierung

Im folgenden wird dieses generelle Vorgehen auf die Einträge der Logfiles angewandt.

2.2.1 Segmentierung (bestimmung der relevanten felder)

In dem Schritt der Segmentierung wollen wir feststellen, welche Informationen aus den Logfiles für die Fragestellung relevant sind. 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 den selben Effekt: man wird auf eine (evtl. vorgefilterte) Seite weiter geleitet. 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 wollen und die Server Antworten irrelevant sind. Demnach sind wir nur an Einträgen interessiert, die in dem Feld [HTTP Request] aus Tabelle 1 mit *Incoming Request* anfangen. Dadurch haben wir die Menge an Einträgen, die wir betrachten wollen schonmal 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.a
*widget=LiquidityByBanksWidgetContent&banks=RpBLVhy85c5u4nGKeISH0A&conversationContext=3&_ns_*
9ab1-472d-bcb0-b6771837ffaa3&_nc_=
Der Name des Widgets ist in der URL erkennbar, wie man hier rot markiert erkennen kann.
Neben dem HTTP Request sind diese weiteren Felder von Relevanz:

Timestamp

SessionID

UserID

2.2.2 Feature extraction (transformation der daten)

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 extrahiert werden, vielmehr werden die Daten in eine bestimmte Struktur transfomiert. Diese Struktur besteht aus drei Mengen, die Menge an usern, die Menge an Session ID's und die Menge an Widgets. Diese Struktur wird von nun an session entity genannt. Eine session entity hat eine eindeutige user ID, eine eindeutige session id und eine Menge an widgets, die in dieser session benutzt wurden. Mit anderen Worten, beinhaltet eine session entity die Informationen welche Widgets in einer session benutzt wurden. Abbildung X soll diesen Zusammenhang verdeutlichen.
Zusätzlich wird zu der Information, welche Widgets benutzt wurden noch festgehalten, zu welchem Zeitpunklt bzw. in welcher Reihenfolge sie benutzt wurden.

2.2.3 Klassifizierung

3 Umsetzung

3.1 Datengenerierung

Die zur Analyse genutzten Daten sind keine tatsächlichen Kundendaten, sondern wurden zum Zweck dieser Abschlussarbeit generiert. Zur Generierung der Daten wurde ein Python Script benutzt. Das war notwendig, da in der aktuellen Version des IFP's die Widgetnamen nicht festgehalten werden. Dies war notwendig, da man in der aktuellen Implementierung nicht alle Widgets anhand der Logdateien erkennen kann. Selbst bei einer schnellen Auslieferung, wäre der Zeitraum zu klein, um genügend Kundendaten zu sammeln. Mithilfe eines Python und Bash Scripts wurden zufällig generierte Logfiles für einen Zeitraum von X generiert.

3.2 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 von 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:

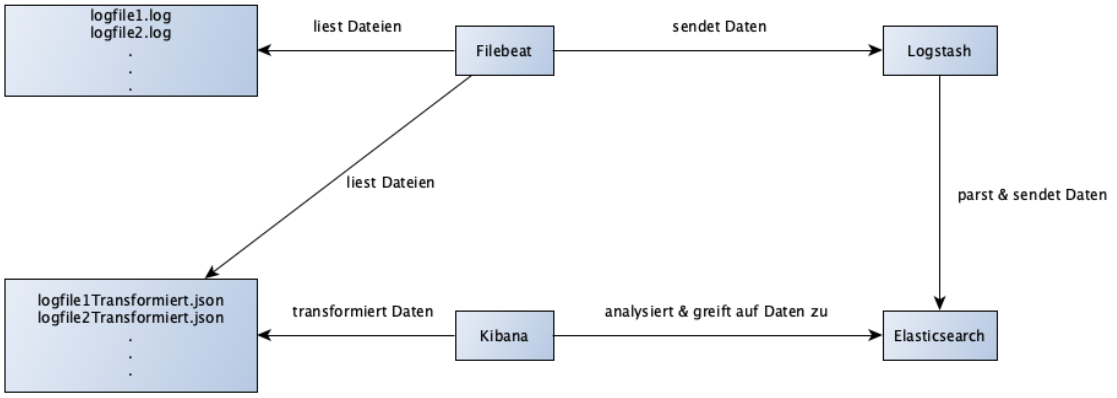

Abbildung 1: ELK Workflow

3.2.1 Filebeat

Mit Filebeat können Dateien zeilenweise gelesen werden. In der Konfigurationsdatei von Filebeat gibt man an, wo und nach welchen Dateien 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 Dateien sendet Filebeat an Logstash

weiter. Dabei merkt sich Filebeat welche Dateien schon gelesen wurden und welche Zeilen in diesen Dateien gelesen wurden. Das bedeutet ebenfalls, wenn eine komplett neue Datei oder eine neue Zeile in einer alten Datei auftaucht, merkt Filebeat das und liest es und sendet es wieder der an logstash. Damit ist die Echtzeitanalyse der Daten gewährleistet.

3.2.2 Logstash

Die Aufgabe von Logstash ist es die Daten, von Filebeat zu empfangen und zu verarbeiten. Die empfangenen Daten befinden sich in einem rohen Zustand und werden mithilfe von mehreren Filter Plug-ins verarbeitet. Nachdem die Daten verarbeitet wurden sendet Logstash sie weiter an Elasticsearch.

Grok Filter Plugin

Für die Logdateien wurde das Grok Filter Plug-in verwendet. In dem Grok-Filter gibt man einen regulären Ausdruck an, mitdem 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 (wie z.B. *LOGLEVEL, TIMESTAMP_ISO8601, SPACE, URIPATH*, uvm.) kann man auch eigene reguläre Ausdrücke definieren. Wie schon in Kapitel 2.2.1 beschrieben, sollen die Daten segmentiert werden, was mit dem Grok-Filter realisiert werden kann. Z.B. kann man in dem regulären Ausdruck an der richtigen Stelle *Incoming Request* setzen. Dadurch würden alle outgoing responses nicht matchen und auch nicht gespeichert werden. Außerdem kann man Daten matchen, aber keinem Feld zuweisen, wodurch sie auch nicht gespeichert werden. Die vollständige Konfiguration für die Logdateien sieht folgendermaßen aus:

```
1  if [log][file][path] =~ "\S+session\S+" {
2          grok{
3              pattern_definitions => {
4                  "REQUEST" => "(Outgoing response|Incoming
                       request)"
5                  "REQUEST_TYPE" => "(POST|GET)"
6              }
7              match => {"message" => [
8                  '^%{TIMESTAMP_ISO8601:zeit}%{SPACE}%{LOGLEVEL}%{
                       SPACE}\[%{NOTSPACE:sessionid}\]\t\[%{NOTSPACE
                       }\]\t\[%{NOTSPACE:userid}\]\t\[%{NOTSPACE} %{
                       NOTSPACE}\]\t%{REQUEST:request}:%{SPACE}%{
                       REQUEST_TYPE:request_type}%{SPACE}%{URIPATH:
                       url}%{GREEDYDATA:stuff}',
9                  '^%{TIMESTAMP_ISO8601:zeit}%{GREEDYDATA:entry}'
10                 ]
11             }
12             remove_field => ['message']
13         }
14
15         if [url] !~ "\S+/rest" {
16             drop{ }
17         }
18
19          mutate {
20             gsub => [ "url", "\S+/rest", ""]
21             remove_field => ['host','agent','@version','ecs','
                   version']
22         }
23
24         date {
25             match => [ "zeit", "ISO8601" ]
26             target => "@timestamp"
27                 }
28
29         if [request] == "Outgoing response"{
30             drop { }
31         }
32
33         if [url] =~ "/fx/\S+" {
34             mutate {
35                 gsub => ["url","/fx/\S+","/fx"]
36             }
37         }
```

Listing 1: Sessionlog Filter

Ruby Filter Plugin

Das Ruby Filter Plugin wird benutzt, um transformierte Daten in Elasticsearch zu speichern. An dieser Stelle sei angemerkt, dass Elasticsearch zwar eine experimentelle *transform* Funktion besitzt, diese aber für unseren Anwendungsfall (noch) nicht passend implementiert ist (Elastic, 2020). Deshalb wurde ein Ruby Skript entwickelt, das diese Funktion ergänzt.

Die Daten, die nach dem Gedanken aus 2.2.2 transformiert wurden, werden in einer JSON Datei gespeichert. Wie genau diese Daten aussehen bzw. wie sie zustande kommen wird in Kapitel 3.2.4 genaue erläutert. Zum besseren Verständis, wie das Ruby Skript arbeitet, wird an dieser Stelle daher die Datenstruktur abstrakt beschrieben. Im Prinzip sind die Daten in 3 Ebenen aufgeteilt:

1. UserIDs
2. SessionIDs
3. Widgets

Dabei kann man die jeweils eine Ebene als ein Array verstehen. So besteht die oberste Ebene aus einem Array, das mit UserIDs gefüllt ist. Zu der UserID wird zusätzlich eine Array gespeichert, das aus SessionIDs besteht, die zu der UserID gehören. Zu diesen SessionIDs wird schließflich ebenfalls ein Array verknüpft, in dem festgehalten wird, welche Widgets in der Sesseion genutzt wurden und wie oft. Nun durchläuft das Skript also die beschriebenen Array Ebenen und speichert die gegebenen Daten in passende Felder, die schließlich an Elasticsearch gesendet und dort in einem passenden Index gespeichert werden.

3.2.3 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 seperat indiziert. Das bedeutet z.B., dass alle Einträge in den Logfiles vom 01.04.2020 und vom 02.04.2020 in eigenen Indizes gespeichert werden.

3.2.4 Kibana

Kibana ist ein Tool, um die Daten, die in Elasticsearch gespeichert sind, zu visualisieren bzw. zu analysieren. Dabei liefert Kibana

4 Auswertung

5 Zusammenfassung

5.1 Fazit

5.2 Zukünftige Anwendungsmöglichkeiten

Literatur

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

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

Abbildungsverzeichnis

Tabellenverzeichnis

Listings