Ausarbeitung/Dokumentation zu unserem Projekt: ScrapeBee

Fach: Textmining

Erstellt von: Thomas Burkart (27961) und Fabian Wenzel (27997)

Wirtschaftsinformatik & Digitale Medien

6. Semester

Hochschule der Medien Stuttgart

Professor: Prof. Dr. Kai Eckert

Abgabe 31.07.2016

Inhaltsverzeichnis

[Einleitung 3](#_Toc457747857)

[Zielsetzung 3](#_Toc457747858)

[Zielsetzung für das Textmining Backend 3](#_Toc457747859)

[Programm-Installation 5](#_Toc457747860)

[Programm-Ausführung 5](#_Toc457747861)

[Datenfluss-Modell 7](#_Toc457747862)

[Aktueller Stand 9](#_Toc457747863)

[Verwendete Technologien 9](#_Toc457747864)

[Textmining Aspekte 10](#_Toc457747865)

[Tokenization 10](#_Toc457747866)

[Location-Identifier 11](#_Toc457747867)

[Spracherkennung 12](#_Toc457747868)

[Content-Analyse 13](#_Toc457747869)

[Prüfung der Artikel auf Gleichheit (Article Similarity) 14](#_Toc457747870)

[Sentiment-Analyse 14](#_Toc457747871)

[Keywords-Extraktion 15](#_Toc457747872)

[Probleme & Hindernisse im Projektverlauf 15](#_Toc457747873)

[Verbesserungen 15](#_Toc457747874)

[Selbstreflektion und Feedback an den Kurs 16](#_Toc457747875)

# Einleitung

Diese Ausarbeitung beinhaltet sowohl eine Dokumentation des Projekts, die aufgetretenen Probleme als auch eine Anleitung des Programms. Unsere Ziele die wir uns gesetzt haben, welche wir erfolgreich erreicht haben und welche nicht. Darunter fällt auch die Erläuterung der Probleme die dabei aufgetreten sind und wie wir diese Probleme gelöst haben. Zu dem dient diese Ausarbeitung als Übersicht über die verwendeten Technologien und den Datenfluss innerhalb unseres Programms. Da wir die Kurse Datenintegration und Textmining besucht haben, liegt der Fokus in dieser Ausarbeitung auf den Aspekten von Textmining.

# Zielsetzung

Unsere erste Idee war, verschiedene News-Seiten und deren Artikel zu vergleichen. Zudem wollten wir eine Methode entwickeln, die verdeutlichen soll, wie Relevant der jeweilige Artikel im Vergleich zu den anderen Artikeln ist. Dabei wollten wir eine Visualisierung der tagesaktuellen Nachrichten entwickeln, mit Hilfe einer Treemap. Neben der Treemap wollten wir eine Google-Map Anzeige zu den Orten, die in den Artikeln vorkommen und die heiß diskutiert werden. Außerdem sollen folgende Fragen am Ende mit Hilfe des Projekts beantwortet werden.

- Zu welchen Themen gibt es die meisten Artikel?  
- Wie ähnlich sind sich die Artikel vom Inhalt?  
- Worüber wird am meisten auf Twitter diskutiert?  
- Wie ist die Stimmung eines Artikels im Vergleich  
 zu anderen Artikeln desselben Themas?  
- Wo finden die Ereignisse statt?

Im Verlaufe des Projekts hat sich auch ein neues Ziel entwickelt: Das Vergleichen von verschiedenen Tools, welche Anbiete und welche Tools passen am besten zu unserem Vorhaben?

## Zielsetzung für das Textmining Backend

Im Folgenden werden die Ziele definiert die wir uns am Anfang des Projektes gesetzt haben für den Bereich Textmining. Im Verlaufe des Projekts wurden nicht alle Ziele genauso umgesetzt, diese werden entsprechend kursiv hervorgehoben.

Allgemein

* Server der über Socket angesprochen werden kann und Text verarbeitet, analysiert und vergleicht

Funktionen - Planung

* Service erhält Link *ODER* Website Content (HTML) und verarbeitet diesen
  + Ergebnis/Rückgabe sollte JSON sein mit UniqueID des Artikels, Keywords zu Content, Ort der Geschehnisse (=> für weitere Abfragen bei Twitter)
  + Beispiel:

     { uniqueId: 123,

   keywords: [ “TTIP”, “Leak”, “USA”, “EU”, “Dokumente” ],

       Location: “USA;EU” }

* *Service speichert alle verarbeiteten Inhalte in MySQL Datenbank (Wir haben bereits auch Code für die Speicherung von Artikeln in einer MySQL Datenbank, jedoch gab es noch ein paar Bugs)*

        Tabelle Artikel:  
        Spalten: uniqueId | link/url | title | keywords | location | language | mood

* Service ermittelt Übereinstimmungen, um Artikel mit gleichen Inhalten zu identifizieren (Vektorspacing!)
* Service liefert sortierte Liste mit übereinstimmenden Artikeln und Ähnlichkeitswerten  
  Beispiel:
  + { { keywords: [ “TTIP”, “Leak”, “USA”, “EU”, “Dokumente” ],

    articleUniqueIds: [ { 123, 0.121 }, { 232, 0.153 }, { 158, 0.24 }, { 32, 0.231 } ]

          },

          { keywords: [ “Prügelei”, “Parlament”, “Türkei” ],

    articleUniqueIds: [ { 342, 0.221 },

{ 234, 0.23 } ]

          } }

Tipps

* Tutorial für WebSocket zwischen nodeJS + Java:

<http://mrbool.com/communicating-node-js-and-java-via-sockets/33819>

* Nochmal ein Tutorial von NodeJs und Websockets:  
  <https://vectortowns.wordpress.com/2014/04/19/first-communication-between-java-and-node-js/>
* in diesem Zusammenhang noch ein weiterer Beispiel Code von stackoverflow :<http://stackoverflow.com/questions/8407460/sending-data-from-node-js-to-java-using-sockets>

Text Extraction:  
https://github.com/kohlschutter/boilerpipe

# Programm-Installation

Unseren Quellcode findet man auch auf github.com unter folgendem Link:   
<https://github.com/FabianWenzel/ScrapeBee.git>

Um das Programm zum Laufen zu bringen werden folgende Komponenten benötigt:

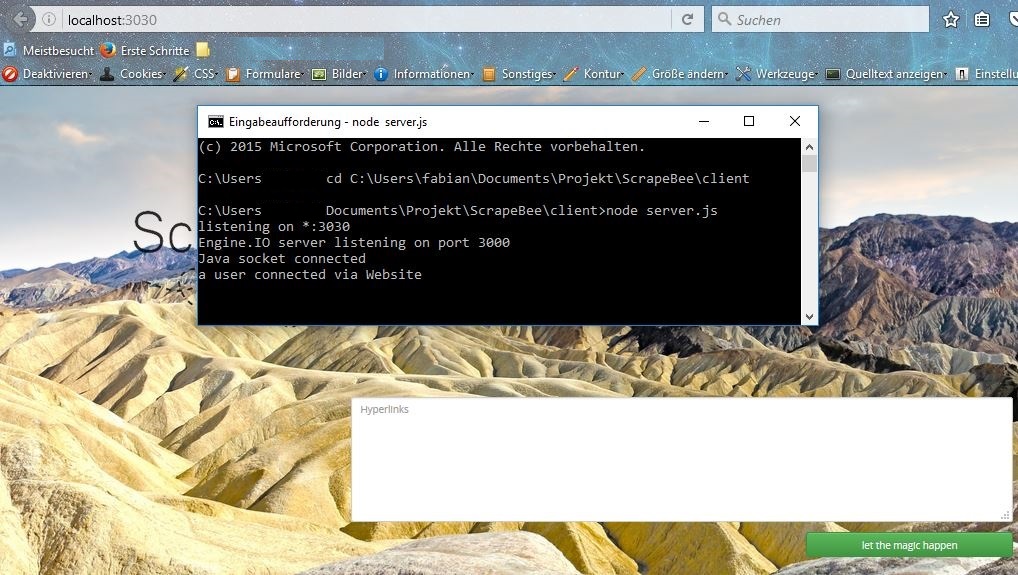
* Eine **Entwicklungsumgebung** wie *Eclipse* oder *Intellij IDEA* (Wir haben *Eclipse* verwendet und würden auch empfehlen *Eclipse* für unser Projekt zu verwenden, da es bei *Eclipse* keine Probleme gegeben hat bei der Einbindung von externen JAR-Files).
* Das **Framework *Node.js*** wird gebraucht für den Webserver und das Crawlern der Newsseiten.
* Einen **Browser** wie *Google-Chrome* oder *Mozilla-Firefox* + aktive Internetverbindung.
* Auf Node.js Seite, benutzen wir verschiedene Module, welche vorher installiert werden müssen (zum Beispiel über den Konsolenbefehl: npm install „Modulname“:   
  - *express* (für den Webserver)  
  - *fs* (File System)  
  - *engine*.*io* (Socket-verbindung)  
  - *socket*.*io* (baut auf engine.io auf)  
  - *net*  
  - *sync*-*request*  
  - *xml2js*
* Unter dem Ordner „lib“ findet man alle JAR-Files die benötigt werden und gegebenenfalls eingebunden werden müssen.
* Wenn das Programm in Eclipse geladen wurde, muss man in der Klasse SystemNames.java noch die Verzeichnis-Pfade für die Classifier anpassen (…//lib//standford//classifiers) Zeile 23,24
* In den Klassen language.java & Sentiment.java muss man noch den Key für die-Yandex API und für die IBM Alchemy-API eintragen

# Programm-Ausführung

Um das Programm auszuführen benötigt man eine Verbindung zwischen dem Node-js Server und unserem Java-BackendService. Als erstes muss der Anwender den Webserver auf Node.js-Seite starten. Dafür öffnet man die Konsole(Eingabeaufforderung) und navigiert in das Verzeichnis in dem die server.js Datei liegt:  
  
„cd …//ScrapeBee//client/“

Dann mit *node server.js* und Eingabetaste, den Server starten - kurz warten bis die 2 Meldungen auf der Konsole erscheinen „*listening on \*:3030*“ und „*Engine.IO server listening on port 3000*“. Unser Webserver lauscht nun auf den Port 3030.  
  
Als nächstes muss man die Klasse *BackendService* als Applikation ausführen:  
(Eclipse 🡪 rechtsklick auf JavaBackend.java 🡪 run as 🡪 Java Application).  
  
In der Konsole dürfte nun die Meldung „*Java socket connected*“ erscheinen. Unser Node Server ist nun mit dem Java-Service verbunden.

Nun öffnet der Benutzer seinen Browser und gibt in das Link-Feld „*localhost:3030*“ ein. In der Konsole müsste eine Meldung kommen mit „*a user connected via Website*“ (siehe Abbildung 1).

  
Abbildung 1: Ausgabe auf der Konsole

Durch einen Klick auf den grünen Button „*let the magic happen*“ startet der eigentliche Vorgang unseres Programms. Auf der Konsole und auf der Website kann man den aktuellen Fortschritt/Status verfolgen. In diesem Schritt finden all die Analysen & der Crawlvorgang statt auf die später nochmals genauer eingegangen wird. Wenn das Programm durchgelaufen ist und somit fertig ist, wird eine Treemap ausgegeben. Zu sehen sind die Kategorien und die einzelnen Artikel. Was die Größe und Farben und Symbole zu bedeuten haben, werden ebenfalls später noch erklärt (Textmining-Aspekte). Im nächsten Kapitel wird der Datenfluss bildlich vorgestellt. Er beschreibt nochmals die Programmausführung anhand von Grafiken.

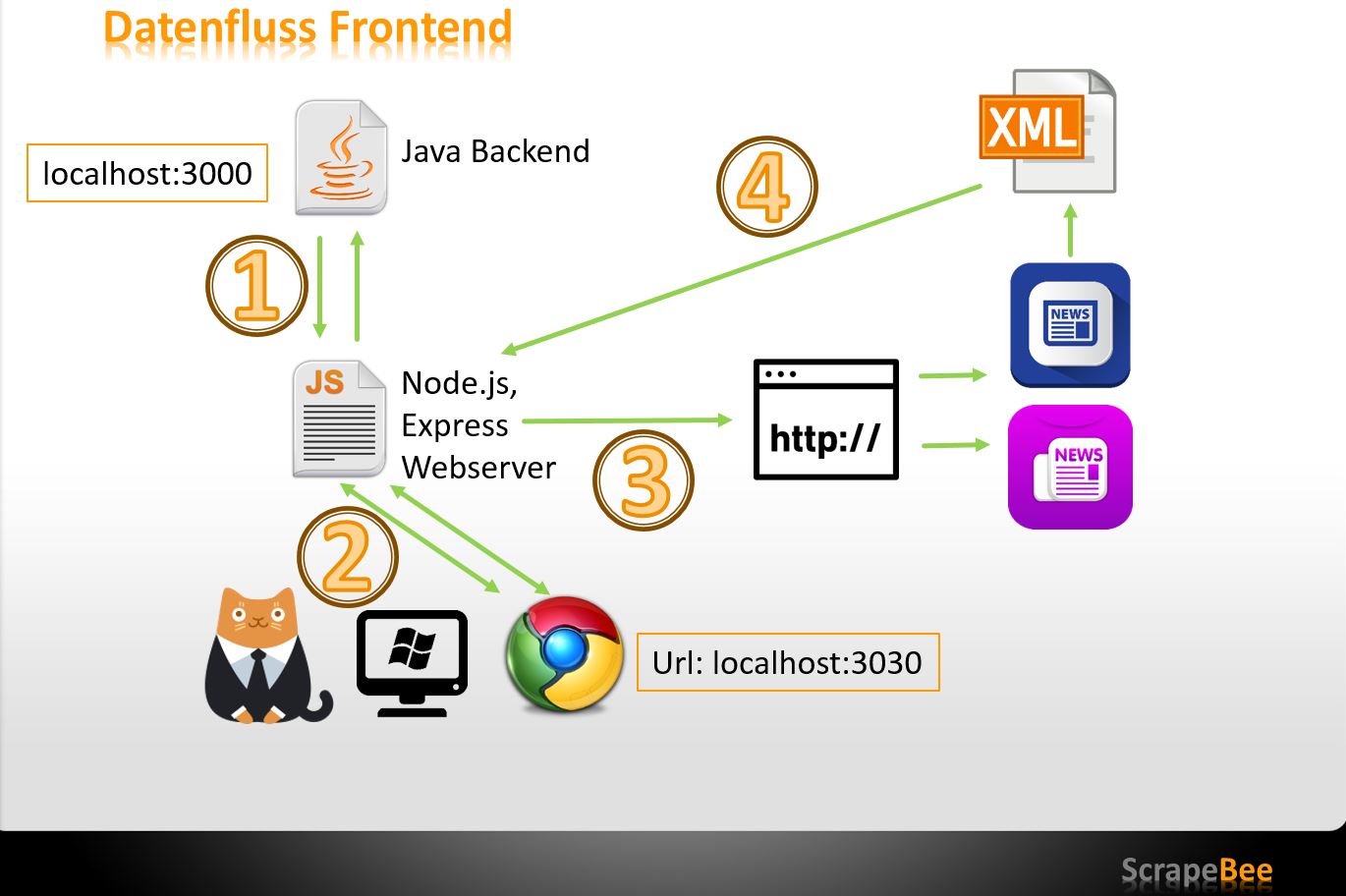
# Datenfluss-Modell

In den 2 folgenden Abbildungen sieht man wohin die Daten „fließen“.  
  
Schritt 1: Verbindung zwischen Node.js Server und Java Backend herstellen.

Schritt 2: Benutzer öffnet Browser und gibt als Adresse localhost:3030 ein.

Schritt 3: Hauptteil des Programms wird ausgeführt(Crawler-Vorgang) RSS-Links werden abgerufen

Schritt 4: Zurück bekommen wir XML.

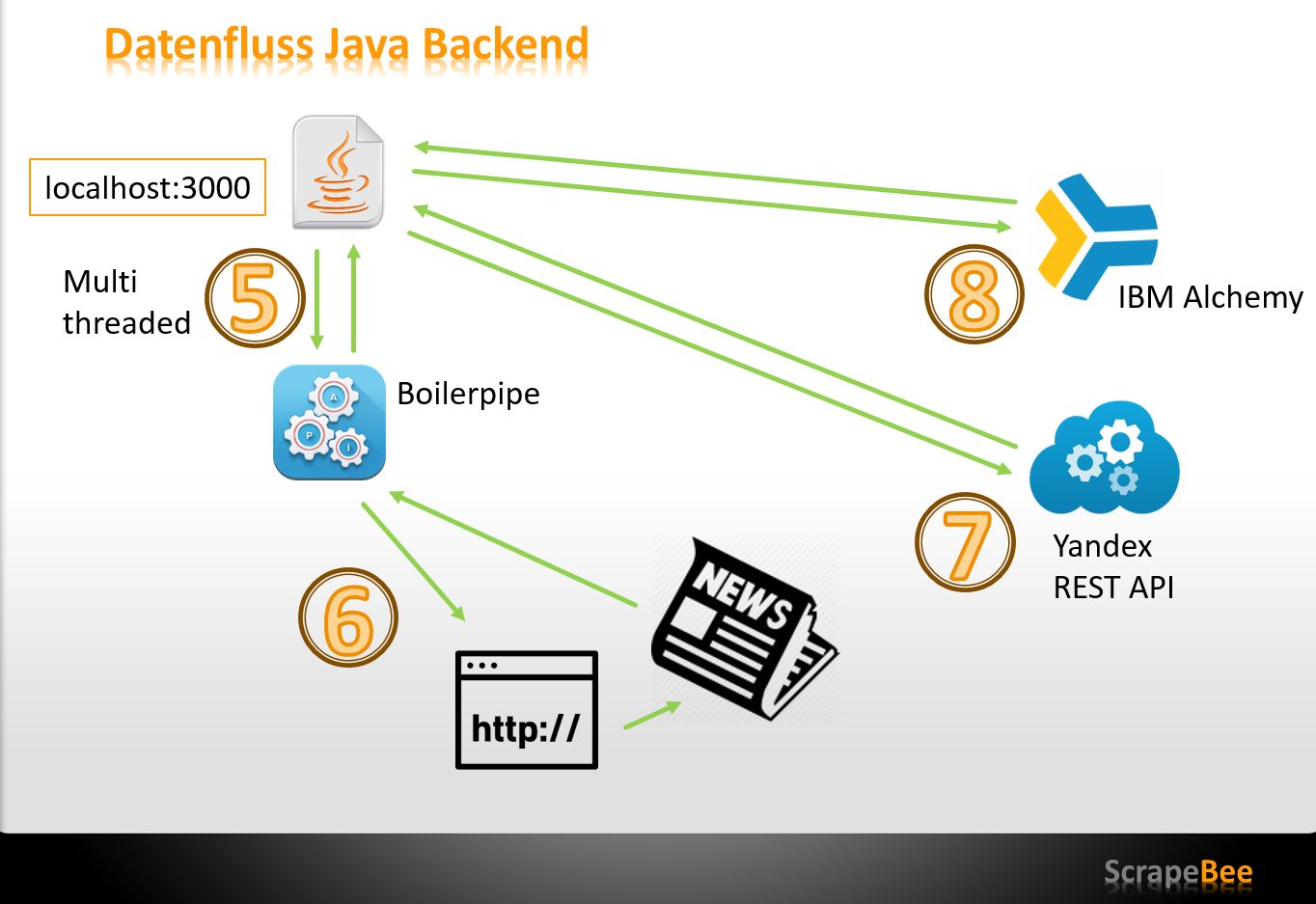
  
Abbildung 2: Datenfluss Schritte 1 bis 4.

Schritt 5: Node.js Server übergibt die Links an das Java Backend, Java Backend startet den Prozess für die Verarbeitung der Links (Boilerpipe zur Extraktion von reinem Text aus einem HTML Dokument)

Schritt 5: Boilerpipe extrahiert den Content aus einem HTML Dokument, zurück kommt reiner Artikel Text ohne Werbung & Social Media Inhalte.

Schritt 7: Mithilfe der Yandex API wird erkannt, in welcher Sprache der Text geschrieben wurde (Spracherkennungsprozess)

Schritt 8: Sentiment Analyse wird durchgeführt durch das Abrufen der von IBM Alchemy bereitgestellten API.

  
Abbildung 3: Datenfluss Schritte 4 bis 8.

Danach kommt noch der Vergleich auf Ähnlichkeit der Newsartikel, die Extraktion der Keywords und das Einteilen in Kategorien, sowie die Aufbereitung für die Treemap. Diese Funktionen werden in dem Klassen-Kapitel genauer erläutert.

# Aktueller Stand

Bisher konnten wir fast all unsere Ziele erreichen☺. Bis auf die Einbindung einer Karte zur Veranschaulich der Locations, die wir bereits aus den Texten herausfiltern können, haben wir soweit alles hinbekommen. Die Locations bekommen wir bereits aus den News-Artikeln, man müsste noch die Google-Maps-API anzapfen und versuchen mithilfe von d3js es auf unserer Website anzuzeigen. In der folgenden Abbildung sieht man noch eine kleine Übersicht aus was unser Quellcode besteht bzw. auf wie viele APIs wir zugreifen.

Abbildung 4: Quellcode Aufbau

# Verwendete Technologien

|  |  |  |
| --- | --- | --- |
| **Technologie** | **Hersteller** | **Funktion** |
| **Boilerpipe** | **github.com/kohlschutter** | **Volltext Extraktion von Webseiten** |
| **REST API – JSON Resp.** | **Yandex** | **Spracherkennung von Texten** |
| **REST API – JSON Resp.** | **Yandex** | **Übersetzung von Texten** |
| **Bluemix – Alchemy API** | **IBM** | **Sentiment Analyse** |
| **Java API** | **Stanford NLP** | **Location Identifier** |
| **Java API** | **Stanford NLP** | **Tokenizer** |
| **MySQL Datenbank** | **Oracle** | **Speichern der bereits extrahierten Artikel / Eigene Spracherkennung** |
| **GSON API** | **Google** | **JSON Verarbeitung** |
| **Engine.io** | **github.com/socketio** | **Kommunikation zu Node.js** |
| **Socket.io** | **github.com/socketio** | **Kommunikation zu Front-End** |
| **Node.js Express** | **Stiftung Node.js** | **Webserver** |
| **Treemap** | **D3JS** | **Visualisierung** |

Abbildung 5: Übersicht der verwendeten Technologien

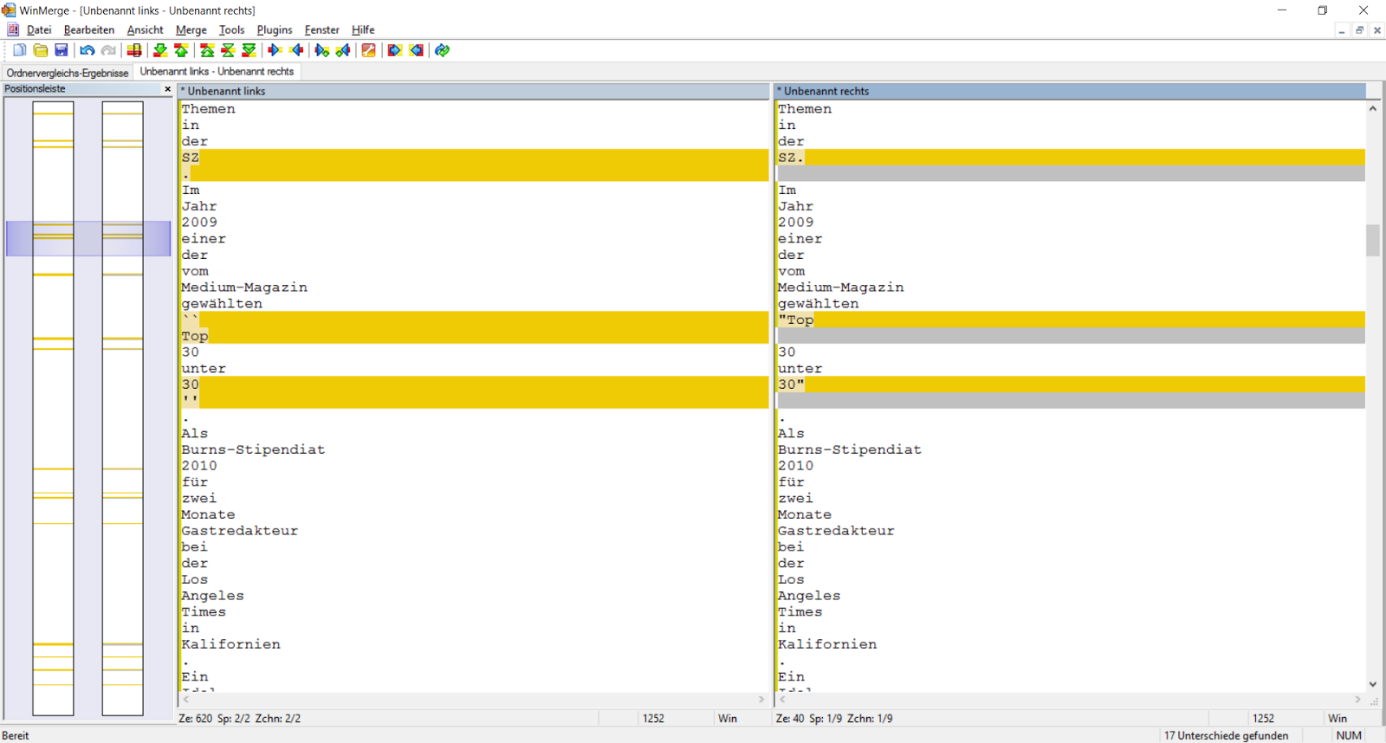
Bei den verwendeten Technologien/Tools wurde im Bereich Textmining, vor allem mit Java gearbeitet. Alle relevanten Textminingaspekte findet man dort wieder in unseren Klassen.

# Textmining Aspekte

In einem anderen Wahlfach namens Analytische Informationswerkzeuge bei Herrn Prof. Meth, gab es eine Vorlesung zum Thema Textmining. Die Definition von Textmining haben wir wie folgt kennengelernt: „Textmining, auch bekannt als Text-Datamining ist ein Algorithmen-basierendes Verfahren zur neuen Erkenntnisgewinnung aus unstrukturierten Text Dokumenten.“ Wir lernten auch die 4-Grund-Schritte des Textmining Prozesses: Selektierung, Vorbereitung, Umwandlung und Datamining. Vor allem bei dem Schritt Vorbereitung gab es viele Übereinstimmungen mit den Themen, die wir bei Herrn Prof. Eckert in dem Fach Textmining besprochen haben. Prozess-Schritte bei der Vorbereitung: Tokenization, Normalisierung, Tagging, Stemming, Entfernung.

Tokenization

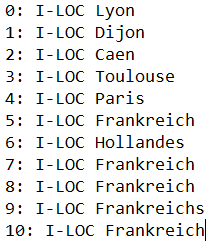
Darunter versteht man die Segmentierung des Textes in einzelne Teile.   
Für unser Programm haben wir uns für das Tool von Stanford NLP entschieden, da es im Vergleich zu Open-NLP, etwas besser zu unserem Vorhaben gepasst hat (siehe Bild). Links Stanford, Rechts Open-NLP. Open-NLP trennt zum Beispiel das Hochkomma vor dem Wort Top nicht.

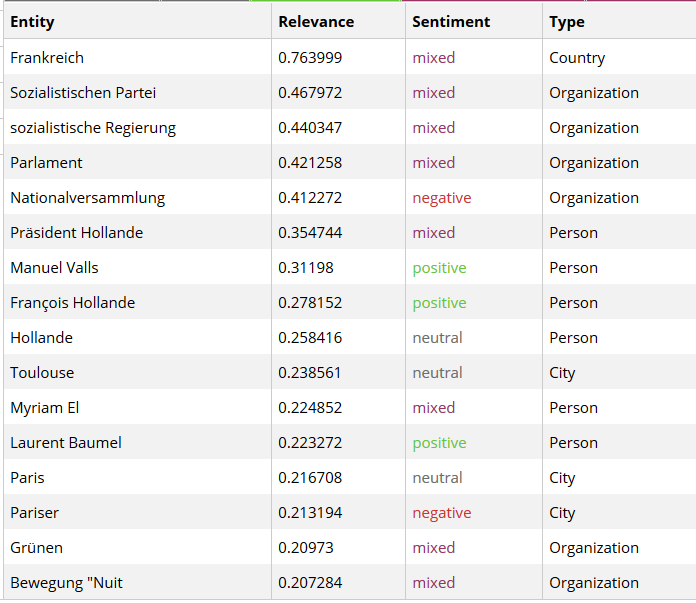
  
Abbildung 6: Vergleich zwischen Tokenizer von Stanford (links) Und Open-NLP (rechts)

Die Prozessschritte Normalisierung, Tagging, Stemming und Entfernung haben wir bei den Texten nicht anwenden müssen, aber bei der Extraktion der Schlüsselwörter wurden diese Methoden angewendet (dazu siehe Kapitel zu Keyword Extraktion).

## Location-Identifier

Um die Orte eines Textes herauszufinden benutzen wir ebenfalls das Tool von Stanford NLP, da es wie auch bei der Tokenization im Test gegenüber dem Tool von Open-NLP besser abgeschnitten hat. Open-NLP hat hierbei teilweise normale Namen von Menschen als Ort interpretiert, anders das Tool von Stanford-NLP, welches die im Text gefundenen Orte mit I-LOC kennzeichnet. Derzeit können wir uns die Orte nur auf der Konsole ausgeben lassen. Geplant war noch, dass wir die Google-API ansprechen und die Orte auf einer Karte anzeigen lassen wollen (z.B.: via Popup Einblendung).

  
Abbildung 7: Ergebnis von Stanford-Location-Identifier

  
Abbildung 8: Alchemy Bluemix Auswertung eines Textes mit Ortsangaben

## Spracherkennung

Anfangs haben wir einen eigenen Code entwickelt um die Sprache eines Textes zu identifizieren. Entwicklung der Spracherkennung eines Textes**:**

Es wurden Methoden in Java geschrieben um Wikipedia Artikel in unterschiedlichen Sprachen einzulesen. Dazu wurde die Wikipedia API „[https://**de**.wikipedia.org/w/api.php?action=query&prop=extracts&explaintext&titles=**key**&format=json&formatversion=2](https://de.wikipedia.org/w/api.php?action=query&prop=extracts&explaintext&titles=key&format=json&formatversion=2)“ abgefragt. In der URL wurde **de** durch den entsprechenden Ländercode ersetzt und der **Key** durch unterschiedliche Wörter. Anschließend lieferte die API eine Rückgabe in JSON Format, entweder mit dem Hinweis auf einen fehlenden Artikel oder den Artikel Text. Der Artikeltext wurde aufgeteilt in einzelne Wörter. Diese wurden der Datenbank hinzugefügt, allerdings nur wenn sie nicht bereits vorhanden sind. Falls sie bereits vorhanden sind, wird das Vorkommnis des Wortes in Verbindung mit der Sprache um eins hochgezählt. Der abgefragte Wikipedia Artikel wurde auch in der Datenbank vermerkt, damit eine nochmalige Abfrage ausgeschlossen werden kann.

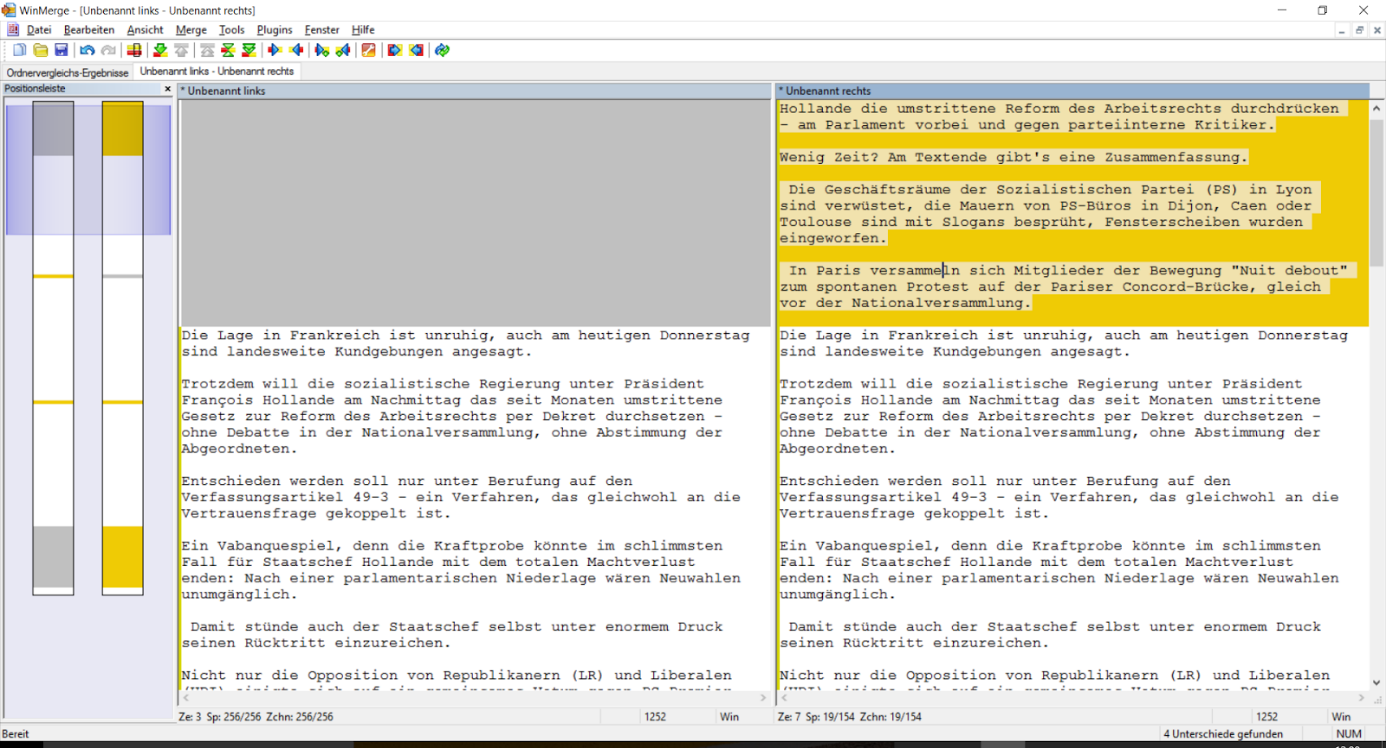
Nun benutzen wir die REST API von Yandex um die Sprache eines Textes zu erkennen. Der Vorteil liegt darin, dass die API viel schneller ist als unser eigener Code. Die Yandex API verfügt auch über eine Übersetzerfunktion, die wir bereits implementiert haben, aber hat bisher noch keine Verwendung gefunden.

## Content-Analyse

Nachdem die Links von dem Node-Server am Java-Backend angekommen sind, werden die Links nacheinander abgearbeitet. Dabei wird der jeweilige Link von Boilerpipe-Klasse verarbeitet. Boilerpipe bekommt als Input den kompletten HTML Code der Website des Artikels und filtert dann den reinen Text heraus. Unter reinem Text versteht sich hierbei der Artikel-Content. In dem folgenden Bild ein Beispiel mit einem Spiegel-Online-Artikel. Boilerpipe lässt alles Überflüssige wie Social-Media und Werbung weg und zieht nur den Text des Artikels.

  
Abbildung 9: Artikelansicht eines Spiegel-Online-Artikels

Für die Content-Analyse wurden mehrere Tools getestet: Boilerpipe, IBM Alchemy, Unfluff & Article Extractor(Node Module). Die Node-Module haben im Vergleich sehr schlecht abgeschnitten, weil dort viel Text verloren ging. Auch bei dem Tool von IBM gab es Lücken, wie im folgenden Bild zu sehen ist.

  
Abbildung 10: Vergleich zwischen IBM Alchemy (links) und Boilerpipe (rechts).

## Prüfung der Artikel auf Gleichheit (Article Similarity)

Die Artikel der Newsseiten werden miteinander auf Ähnlichkeit geprüft mit Hilfe des Vector-Spacing und dem TF, IDF – Maß.   
Idee: die wichtigsten Terme (bzgl. einer Recherche) sind diejenigen, die häufig in individuellen Dokumenten auftreten und selten im restlichen Dokumentenbestand.  
Das Tf-idf-Maß wird im Vectorspacing zur Beurteilung der Relevanz von Termen in Dokumenten einer Dokumentenkollektion eingesetzt. Bei uns also in den Artikeln. Mit der so errechneten Gewichtung eines Wortes bezüglich des Dokuments, in welchem es enthalten ist, können Dokumente als Suchtreffer einer wortbasierten Suche besser in der Trefferliste angeordnet werden, als es beispielsweise über die Termfrequenz allein möglich wäre.   
  
Formel: IDF(i) = log2 (N /n) + 1   
  
Ab einem Wert von 0,2 gelten bei unserem Programm, 2 Artikel als ähnlich. Die Ähnlichkeit wird in der Treemap mittels der Farbentransparenz dargestellt (desto transparenter die Farbe, desto „unähnlicher“ sind sich die Artikel“).

## Sentiment-Analyse

Mit Hilfe der Sentiment-Analyse wollten wir feststellen, wie interpretieren die Tools die Texte der Autoren der Artikel, also ist die geäußerte Haltung eher positiv oder negativ? Die Häufigkeiten positiver und negativer Begriffe im analysierten Text werden einander gegenübergestellt und bestimmen die vermutete Haltung. So arbeitet auch die API von IBM-Bluemix für die wir uns hierbei entschieden haben. Die API liefert zu jedem Wort einen bestimmten Wert zwischen -1 (negativ), 0 (neutral) und +1 (positiv). Der Durchschnitt aller Wörter ergibt den Gesamtwert. In der später generierten Treemap wird die Stimmung angezeigt durch ein +, ~ oder – nach dem Titel des Artikels. Dabei mussten wir feststellen, dass manche Artikel zum gleichen Thema andere Stimmungen hervorrufen, somit muss man sich eigentlich fragen ob die Sentiment-Analyse hierbei wirklich sinnvoll ist, da es doch noch einige Probleme zu lösen gilt wie die Interpretation von Sarkasmus bzw. allgemein Fehlinterpretationen.

## Keywords-Extraktion

Bei der Suche nach Schlüsselwörtern in Texten hat bei uns kein einziges Tool, welches wir getestet hatten, gute Ergebnisse geliefert. Stanford, Open-NLP und IBM Bluemix lieferten für uns keine sinnvollen Schlüsselwörter (darunter auch Verben und Adjektive).   
Mittels TF-IDF hätten wir wahrscheinlich bessere Ergebnisse bekommen, jedoch kamen wir zu spät auf diese Idee.

Unser jetziger Ansatz zur Schlüsselwörter-Extraktion bildet sich aus der der Normalisierung des Textes und der Stop-word Entfernung des Titels. Dabei wird als erstes der Text und der Titel in Kleinbuchstaben umgewandelt und im Titel werden dann Stopp-Wörter wie Artikel (Der, Die, Das) und andere Wörter wie „und, dann, sind, auch, er, in etc.“ entfernt. Anschließend werden die Wörter des Titels mit den Wörtern im Text vergleichen (die Häufigkeit). Die Wörter die am häufigsten im Text wiedergefunden werden, sind nun unsere Keywords. Die Keywords werden dann später in der Suche von Twitter verwendet um Herauszufinden wie viele Tweets es zu diesen Wörtern gibt. Anhand der Anzahl können dann weitere Thesen aufgestellt werden wie umso höher die Anzahl an Tweets, desto höher ist gerade das Interesse bzw. ein Hinweis auf ein sehr aktuelles und spannendes Thema.

# Probleme & Hindernisse im Projektverlauf

* Bei der entwickelten Spracherkennung kam es zu langen Programmlaufzeiten die wir durch den API Aufruf von YANDEX nun umgehen können. Nicht nur die Programmlaufzeit war zu hoch, sondern auch die Entwicklungszeit dafür. Das Durchsuchen von 90 000 Wikipedia-Artikeln und das anschließende Analysieren und Abspeichern in die Datenbank in 3 verschiedenen Sprachen beanspruchte volle 4 Tage.
* Die Keyword-Extraktion mit den Tools hat nicht so gut geklappt wie wir uns das vorgestellt hatten (wie oben bereits beschrieben). Durch TF & IDF hätten wir das umgehen können, jedoch beließen wir es bei unserer jetzigen Lösung mit der Stopp-Wort-Entfernung der Titel und dem Vergleich mit dem Text (Vorkommen im Text).

# Verbesserungen

* TF & IDF für die Keywords
* Google Maps Implementierung

# Selbstreflektion und Feedback an den Kurs

Wir sind mit unserem Projekt mehr als Zufrieden und sind sehr stolz darauf, dass wir fast alle unsere Ziele erreichen konnten. Zudem hat uns dieses Projekt sehr viel Spaß gemacht und wir lernten auch einiges dazu. Wir würden sogar behaupten, wir haben bisher in unserem Studium, in keinem anderen Fach so viel (Nützliches) gelernt wie in den beiden Veranstaltungen Textmining und Datenintegration & Analyse.

Endstand:

