Übungsblatt 3

# Crawling

Für das crawlen der Tweets habe ich die Programmiersprache PHP mit dem Paketverwalter Composer[[1]](#endnote-1) verwendet. Über Composer installierte ich dann das Paket fennb/phirehose, welches den Zugriff auf die Twitter Streaming API, sowie die Authentifizierung über OAuth übernahm.

Mit meinem Programm streamte ich mir dann 100.000 Tweets mit den Feldern: username (Twitter Benutzername), text (Tweetinhalt), coordinates (GPS-Koordinaten), place (Stadt/Ort), country\_code (ISO 3166-1 alpha 2 Code des Landes) und country (Name des Landes). Ich habe bewusst nur 100.000 Tweets gespeichert, um die Laufzeit der verlangten MapReduce-Jobs in Grenzen zu halten.

Diese Daten wurden bereits während des Crawling-Vorgangs in MongoDB in der Datenbank test unter der Collection tweets abgelegt. Das ISO-Kürzel des Landes wird übrigens benötigt, da er absolut eindeutig und standardisiert ist, wohingegen der Ländername von Twitter in der lokalisierten Variante zurückgeliefert wird. So bekommt man für Österreich beispielsweise „Österreich“, „Austria“ oder auch „Autriche“ zurückgeliefert, je nachdem welche Sprache der jeweilige Benutzer eingestellt hat.

**Crawler-Code:**

<?php

**error\_reporting**(**E\_ALL**);

ini\_set("display\_errors", 1);

**require**("vendor/autoload.php");

/\*\*

\* General purpose crawler class for the Twitter streaming API.

\*

\* @author Markus Deutschl <mdeutschl.mmt-m2012@fh-salzburg.ac.at>

\*/

**class** TweetCrawler **extends** OauthPhirehose {

**private** $collection;

**private** $batchSize;

**private** $maxTweets;

**private** $tweetCount = 0;

**private** $tweets = [];

**public** **function** \_\_construct($username, $password, $method = Phirehose::METHOD\_SAMPLE, $format = self::FORMAT\_JSON, $lang = **false**, $db = "test", $collection = "tweets", $maxTweets = 100000, $batchSize = 1000) {

**parent**::\_\_construct($username, $password, $method, $format, $lang);

$this->maxTweets = $maxTweets;

$this->batchSize = $batchSize;

$this->collection = (**new** MongoClient())->selectDB($db)->selectCollection($collection);

}

**public** **function** enqueueStatus($status) {

// Flush the tweets to the database if the batch size is big enough.

**if** (!**empty**($this->tweets) && $this->tweetCount % $this->batchSize === 0 ) {

$this->collection->batchInsert($this->tweets, [ "continueOnError" => **true** ]);

$this->tweets = [];

$this->log("Flushed {$this->tweetCount} of {$this->maxTweets} tweets to the database!");

}

// If we have gathered enough tweets, flush the last tweets to the database and exit.

**if** ($this->tweetCount >= $this->maxTweets) {

**if** (!**empty**($this->tweets)) {

$this->collection->batchInsert($this->tweets, [ "continueOnError" => **true** ]);

}

$this->log("{$this->maxTweets} tweets have been crawled successfully!");

**exit**();

}

// Decode the JSON data to an associative array.

$data = json\_decode($status, **true**);

// Check for all required fields.

**if** (is\_array($data)

&& **isset**($data["user"])

&& **isset**($data["user"]["screen\_name"])

&& **isset**($data["text"])

&& **isset**($data["geo"])

&& **isset**($data["geo"]["coordinates"])

&& **isset**($data["place"])

&& **isset**($data["place"]["name"])

&& **isset**($data["place"]["country\_code"])

&& **isset**($data["place"]["country"])

) {

// Push the tweet data to the internal stack.

$this->tweets[] = [

"user" => $data["user"]["screen\_name"],

"text" => $data["text"],

"coordinates" => $data["geo"]["coordinates"],

"place" => $data["place"]["name"],

"country\_code" => $data["place"]["country\_code"],

"country" => $data["place"]["country"],

];

$this->tweetCount++;

}

}

}

// Check if the configuration file is readable.

**if** (!is\_readable("crawler.ini")) {

**exit**("Please supply a crawler.ini file next to the crawler.php!" . **PHP\_EOL**);

}

// Parse the necessary options.

$options = parse\_ini\_file("crawler.ini");

// Check if we have all options needed.

**if** ($options === **false**

|| !**isset**($options["consumer\_key"])

|| !**isset**($options["consumer\_secret"])

|| !**isset**($options["oauth\_token"])

|| !**isset**($options["consumer\_secret"])

) {

**exit**("Missing configuration options in crawler.ini! Please supply consumer\_key, consumer\_secret, oauth\_token and oauth\_secret." . **PHP\_EOL**);

}

$crawler = **new** TweetCrawler($options["oauth\_token"], $options["oauth\_secret"], Phirehose::METHOD\_FILTER);

// Set consumer key and secret for the Twitter app.

$crawler->consumerKey = $options["consumer\_key"];

$crawler->consumerSecret = $options["consumer\_secret"];

// Only crawl tweets with a geolocation and in the English language.

$crawler->setLocations([[-180, -90, 180, 90]]);

$crawler->setLang("en");

// Finally consume the Tweet stream.

$crawler->consume();

# MongoDB

## Import

Der Import-Schritt fiel weg, da die Collection schon während des Crawling-Vorgangs befüllt wurde.

## Count

MapReduce ist hier etwas fehl am Platz, da MongoDB schon eine Funktion anbietet, um Datensätze zu zählen, die bereits performant ist. Bei MapReduce kann man hier auch nur einen Output-Key verwenden, was wohl eher suboptimal ist.

**MongoDB-Kommando zum Zählen:**

db**.**tweets**.**find**().**count**();**

**MapReduce-Aufruf zum Zählen:**

db**.**tweets**.**mapReduce**(**

**function** **()** **{** emit**(**1**,** 1**);** **},**

**function** **(**key**,** values**)** **{** **return** Array**.**sum**(**values**);** **},**

**{** out**:** **{**inline**:** 1**}** **}**

**).**results**[**0**].**value**;**

## Slang-Übersetzung

Das Slang-Dictionary wurde mittels Regular Expression in ein für MongoDB verständliches Format übersetzt und anschließend in die Collection slang mittels mongoimport eingefügt.

**MapReduce Code:**

**var** slang **=** **{};**

db**.**slang**.**find**().**forEach**(function** **(**elem**)** **{**

slang**[**elem**.**\_id**]** **=** elem**.**value**;**

**});**

db**.**tweets**.**mapReduce**(**

**function** **()** **{**

**this.**text**.**split**(**/[,.\?\!\-\_ ]/**).**forEach**(function** **(**word**)** **{**

**if** **(**word**.**length **>** 0**)** **{**

word **=** word**.**toLowerCase**();**

**if** **(**slang**[**word**])** **{**

slang**[**word**].**split**(**" "**).**forEach**(function** **(**elem**)** **{**

emit**(this.**\_id**,** **{**

word**:** elem**,**

coordinates**:** **this.**coordinates**,**

place**:** **this.**place**,**

country\_code**:** **this.**country\_code

**}**

**);**

**}.**bind**(this));**

**}**

**else** **{**

emit**(this.**\_id**,** **{**

word**:** word**,**

coordinates**:** **this.**coordinates**,**

place**:** **this.**place**,**

country\_code**:** **this.**country\_code

**}**

**);**

**}**

**}**

**}.**bind**(this));**

**},**

**function** **(**key**,** values**)** **{**

**var** words **=** **[];**

values**.**forEach**(function** **(**element**)** **{**

words**.**push**(**element**.**word**);**

**});**

**return** **{**coordinates**:** values**[**0**].**coordinates**,** country\_code**:** values**[**0**].**country\_code**,** place**:** values**[**0**].**place**,** words**:** words**};**

**},**

**{** out**:** "tweets\_translated"**,** scope**:** **{**slang**:** slang**}** **}**

**);**

## Subjectivity Lexicon

Das Format des Lexikons wurde ebenfalls angepasst und in die Collection subjectivity importiert.

1. <http://getcomposer.org/> [↑](#endnote-ref-1)