Übungsblatt 3

1. Crawling

Für das crawlen der Tweets habe ich die Programmiersprache PHP mit dem Paketverwalter Composer[1] verwendet. Über Composer installierte ich dann das Paket `fennb/phirehose`[2], 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 (crawler.php):

```php
<?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);
  }
```

[1] http://getcomposer.org/
[2] https://packagist.org/packages/fennb/phirehose

```
  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();
```

2. MongoDB

a. Import

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

b. 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 (2.js):

```
db.tweets.mapReduce(
  function () { emit(1, 1); },
  function (key, values) { return Array.sum(values); },
  { out: {inline: 1} }
).results[0].value;
```

c. 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 (2.js):

```
var slang = {};
db.slang.find().forEach(function (elem) {
    slang[elem._id] = elem.value;
});
var punctuation = ",;\\.:\\-
_#'\\+\\*´`\\?!\"=\\(\\)\\{\\}\\|\\]\\\\~\\\\\\^„“”";
var re = new RegExp("^[" + punctuation + "]*([^" + punctuation + "]+)[" +
punctuation + "]*$");


db.tweets.mapReduce(
  function () {
    this.text.split(" ").forEach(function (word) {
      if (word.length > 0) {
        word = word.replace(re,"$1");
        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, re: re} }
);
```

d. Subjectivity Lexicon

Das Format des Lexikons wurde ebenfalls angepasst und in die Collection `subjectivity` importiert, diesmal mithilfe von `bash` Pipes und Filtern.

`bash`-Kommandozeile:

```
cat subjclueslen1-HLTEMNLP05.tff | cut -d' ' -f 3,6 | \
sed "s/word1=//" | sed "s/priorpolarity=//" | tr ' ' ',' | \
uniq > lexicon.txt
```

e. Sentiment Analysis

Um Tweets mit leerem Text auszufiltern, wurde dem MapReduce-Aufruf diesmal ein zusätzlicher Query übergeben.

MapReduce Code (2.js):

```
var subj = {};
db.subjectivity.find().forEach(function (elem) {
  subj[elem._id] = elem.rating;
});

var mapping = {
  negative: -1,
  neutral: 0,
  positive: 1
};

db.tweets_translated.mapReduce(
  function () {
    this.value.words.forEach(function (word) {
      if (subj[word]) {
        emit(this._id, {
          rating: mapping[subj[word]],
          coordinates: this.value.coordinates,
          place: this.value.place,
          country_code: this.value.country_code
        });
      }
    }.bind(this));
  },
  function (key, values) {
      rating = 0;
      values.forEach(function (elem) {
          rating += elem.rating;
      });
      return {coordinates: values[0].coordinates, country_code:
values[0].country_code, place: values[0].place, rating: rating};
  },
  { out: "subj_analysis", scope: {subj: subj, mapping: mapping}, query:
{"value.words": {$exists: true}}}
);
```

f. Sentiment Analysis mit Emoticons

Die hier verwendeten Emoticons stammen von der Wikipedia[3] Es wurden hier allerdings nur die Emoticons aus der ersten Tabelle berücksichtigt.

MapReduce Code (2.js):

```
var subj = {};
db.subjectivity.find().forEach(function (elem) {
  subj[elem._id] = elem.rating;
});

var mapping = {
  negative: -1,
  neutral: 0,
  positive: 1
};

var emoticons = {
  positive: [":-
)",":)",":o)",":]",":3",":c)",":>","=]","8)","=)",":}",":^)",":ↄ)",":-
D",":D","8-D","8D","x-D","xD","X-D","XD","=-D","=D","=-3","=3","B^D",":-
))",":*",":^*","( '}{' )",">:P",":-P",":P","X-P","x-p","xp","XP",":-
p",":p","=p",":-Ꝥ",":Ꝥ",":ꝥ",":-ꝥ",":-b",":b","\\o/","<3"],
  negative: [">:[",":-(",":(","",":-c",":c",":-<","",":ↄC",":<",":-
[",":[",":{",";(",":-||",":@",">:(",":'-(",":'(",":'-
)",":')","D:<","D:","D8","D;","D=","DX","v.v","D-':",">:O",":-O",":O","8-
0",">:\\",">:/",":-/",":-.",":/",":\\","=/","=\\",":L","=L",":S",">.<",":-
###..",":###..","<:-|","</3"]
};
```

[3] http://en.wikipedia.org/wiki/List_of_emoticons

```
db.tweets_translated.mapReduce(
  function () {
    this.value.words.forEach(function (word) {
      var rating = mapping.neutral;
      if (subj[word]) {
        rating = mapping[subj[word]];
      }
      else if (emoticons.positive.indexOf(word) !== -1) {
        rating = mapping.positive;
      }
      else if (emoticons.negative.indexOf(word) !== -1) {
        rating = mapping.negative;
      }
      emit(this._id, {
        rating: rating,
        coordinates: this.value.coordinates,
        place: this.value.place,
        country_code: this.value.country_code
      });
    }.bind(this));
  },
  function (key, values) {
      rating = 0;
      values.forEach(function (elem) {
          rating += elem.rating;
      });
      return {coordinates: values[0].coordinates, country_code:
values[0].country_code, place: values[0].place, rating: rating};
  },
  { out: "subj_analysis_emo", scope: {subj: subj, mapping: mapping,
emoticons: emoticons}, query: {"value.words": {$exists: true}}}
);
```

3. Visualisierung

Für die Visualisierung wurden die bereits gesammelten Stimmungswerte der Tweets inklusive Emoticons für Länder aggregiert und der Landesmittelwert mittels MapReduce in MongoDB berechnet. Diese Ergebnisse wurden dann mittels PHP aus der Collection in ein JavaScript-Array eingelesen und mit der Google GeoChart API visualisiert. Dabei kommt eine ganz ansehnliche Grafik heraus, auf der man relativ gut die Stimmungsunterschiede der Länder zum Zeitpunkt des Tweet-Crawlings erkennen kann.

MapReduce Code für die Berechnung des Stimmungswerts pro Land (3.js):

```
db.subj_analysis_emo.mapReduce(
  function () {
    if (this.value.country_code && this.value.country_code.length === 2) {
      emit(this.value.country_code, this.value.rating);
    }
  },
  function (key, values) {
    return Array.sum(values) / values.length;
  },
  { out: "sentiment_country" }
);
```

Darstellung der Ergebnisse mit PHP + HTML + CSS + JS + Google GeoChart API (index.php):

```
<html>
  <head>
    <title>Mood Visualization of Countries</title>
    <meta charset="utf-8">
  </head>
  <style type="text/css">
    #mood-map {
      height: 500px;
      width: 900px;
    }
  </style>
  <body>
    <h1>Mood Visualization of Countries according to Twitter</h1>
    <p>The moods of the countries in the analysis have been
calculated from the contents of roughly 100,000 tweets, crawled on
December, 5th 2013.</p>
    <p>Words and their positive or negative meaning were analyzed
and rated. The final score was aggregated for each country in the
dataset.</p>
    <p>In total 100,000 tweets from 43 countries have been rated and
visualized in the chart below.</p>
    <div id="mood-map"></div>
    <script type='text/javascript'
src='https://www.google.com/jsapi'></script>
    <script type="text/javascript">
      google.load("visualization", "1", {"packages": ["geochart"]});
      google.setOnLoadCallback(drawMoodMap.bind(this));
      var data = [ ["Country", "Mood"]<?php
        $cursor = (new MongoClient())->selectDB("test")-
>selectCollection("sentiment_country")->find();
        foreach ($cursor as $doc) {
          echo ", ['{$doc["_id"]}', {$doc["value"]}]";
        }
      ?> ];

      var min = Infinity;
      var max = 0;
      data.forEach(function (elem) {
        if (elem[1] > max) {
          max = elem[1];
        }
        if (elem[1] < min) {
          min = elem[1];
        }
      });

      function drawMoodMap() {
        var chartData = google.visualization.arrayToDataTable(data);
        var options = {
          colorAxis: {
            colors: [ "red", "white", "green" ],
            values: [ min, 0, max]
          },
          datelessRegionColor: "white"
        };

        var chart = new
google.visualization.GeoChart(document.getElementById("mood-map"));
        chart.draw(chartData, options);
      }
    </script>
  </body>
</html>
```