Big data techniques - practical work

This project is about to get used to the topic of big data and try to get practical knowledge how big data challenges can be achieved by working with different popular frameworks and techniquies.

Requirements

This project has the following dependencies:

- Java (version 1.8.x)
- Docker (version 18.x or greater)

Within the application cycle there are also other tools which comes in use while executing the files:

- Apache Flink (version 1.6.x)
- Elasticsearch (version 6.4.x)
- Kibana (version 6.4.x)

In the Java environment the following dependencies are used and get loaded through maven:

- org.apache.flink.flink-connector-twitter (version 1.6.0)
- org.apache.flink.flink-connector-elasticsearch (version 1.6.0)
- org.elasticsearch.client.transport (version 6.4.0)
- apache commons-lang (version 2.6)
- org.apache.commons.commons-math3 (version 3.6.1)
- com.googlecode.json-simple (version 1.1.1)

All shown commands are executed on macOS, but should also work in a common linux environment.

Setup

In the developing process JetBrains **IntelliJ** is used. It manages to install the needed Maven dependencies. The used Flink dependency also provides an instance of flink which runs out of the application. So no installed version of flink in the system is needed.

The same situation can be find with *Elasticsearch* and *Kibana*. To bypass the need of installing both on bare metal I used the abstraction through **Docker**. For testing you just can use the provided `docker-compose` files in the `env/` folder.

While using Apache Flink version 1.6 in this project, there is also being used the latest version of *Elasticsearch* and *Kibana*. The `docker-compose.latest.yml` file can be used to run this setup. To start the elasticsearch environment use:

```
$ docker-compose -f env/docker-compose.latest.yml up -d
```

When you do not want to install **Docker** on your system, you also can use fully installed instances of *Elasticsearch* and *Kibana*. You then have to change the config parameters in the `src/main/resources/elasticsearch.properties` file to your environment.

Project structure

```
.
├── README.md
├── data [Raw output data from streaming jobs - used by for batch
processing]
├── env [docker-compose file and additional environment data like ES raw
data]
├── output [Output of streaming and batch jobs]
├── pom.xml [Maven project file]
├── src
│   └── main
│       ├── java
│       │   └── de
│       │       └── moritzkanzler
│       │           ├── batch
│       │           ├── comparison
│       │           ├── exception
│       │           ├── filter
│       │           ├── helper
│       │           ├── keyBy
│       │           ├── map
│       │           ├── model
│       │           ├── prediction
│       │           ├── sink
│       │           ├── streaming
│       │           └── utils
│       └── resources [Resources used by the java programs]
├── target [...]
└── utils [Additional non-java programs for small scripting purposes]
    └── json-converter.js
```

Properties

In order you want to play around with the given setup most of the used options are outsourced from the java code into several property files. These files can be found under `src/main/resources'. The goal was to enable a configurable project, where the options also can be changed later on without the need to dive deeper into the code basis. The following table shows the purpose of each file. The different options are explained by comments above them in the property file:

Filename	Task
elasticsearch.properties	This property file holds the settings for the connection to an *elasticsearch* instance, which is used in the result presentation. Per default the settings are configured to work with the elasticsearch docker environment
general.properties	In this property file can be found different options the program uses to configure or define outputs or input resources

Filename	Task
log4j.properties	This is the default property file of the log4j logging framework which is used in this project
twitter.properties	The twitter property file holds the information the flink connector uses to access the twitter api with your own credentials. The credentials can be achieved by register for the Twitter API service:
Other files:	
programs/	In the *programs/* folder can be find several files. Each one is for one metric flow. Normally there can be found some boolean flags to control the output of the program and paticular paths to store it. It is also possible to redefine values from the `general.properties` file.
country-capitals/	In one metric, a mapping happen where language codes get transformed to coordinates of the capitals of several countries. To get these information a external list is used which can be found in this folder. There are two versions, one in JSON format and one in CSV. The program uses the JSON version.

Metrics

1. Sum of retweets per tweet
2. Top 5 Retweets
3. Retweets per country (per hour)
4. Percentage of all retweets per country (per hour)
5. Percentage of retweets regarding all tweets of country (per hour)

1. Retweets in total per time

program file	`TweetRetweetsCountry.java`
property file	`tweetretweetscountry.properties`
window size	15 minutes
total runtime	24 hours
date	*29th Oct 2018 1445* to *30th Oct 2018 1445*
output	Console, Elasticsearch, and CSV
stream (ES index):	twitter-retweets
stream (ES Visualization):	-
stream (CSV):	`output/tweets-retweets.stream.csv`
data:	`data/tweet-retweets.csv`
batch (CSV):	`output/tweet-retweets.batch.csv`

This metric sets the ground for the following metrics and show the amount of retweets happening per given time window.

2. Top X Retweets per time

program file	`TweetRetweetsCountry.java`
property file	`tweetretweetscountry.properties`
window size	15 minutes
total runtime	24 hours
date	*30th Oct 2018 1745* to *30th Oct 2018 1845*
output	Console
stream (ES index):	-
stream (ES Visualization):	-
stream (CSV):	`output/tweet-top-x.stream.csv
data:	`data/tweet-retweets-top-x.csv
batch (CSV):	`output/tweet-top-x.batch.csv

This metrics chooses the 5 most retweeted tweets per given time window.

3. Retweets per country per time

program file	`TweetRetweetsCountry.java`
property file	`tweetretweetscountry.properties`
window size	15 minutes
total runtime	24 hours
date	*9th Oct 2018 1530* to *10th Oct 2018 1530*
output	Console, Elasticsearch, and CSV
stream (ES index):	twitter-retweets-country
stream (ES Visualization):	Country with most retweets
stream (CSV):	`output/tweet-retweets-country-geo.stream.csv`
data:	`data/tweet-country.csv`
batch (CSV):	`output/tweet-retweets-country-geo.batch.csv`

With this metric we can process which countryin a given time window have how much retweets. The country gets chosen by the language of the author of the reposted tweet. Addition to that the processing adds location data of the capital of the given country by using `de.moritzkanzler.utils.Utils.langCodeToCoords()`.

4. Percentage of all retweets per country

program file	`TweetRetweetsCountry.java`
property file	`tweetretweetscountry.properties`
window size	15 minutes
total runtime	24 hours
date	*9th Oct 2018 1530* to *10th Oct 2018 1530*
output	Console, Elasticsearch, and CSV
stream (ES index):	twitter-retweets-country-percentage
stream (ES Visualization):	Percentage of Retweets per country
stream (CSV):	`output/tweet-retweets-country-percentage.stream.csv`
data:	`data/tweet-country.csv`
batch (CSV):	`output/tweet-retweets-country-percentage.batch.csv`

This metric also sums the amount of retweets per country per time frame and finally calculate a share between these amounts through out the countries. This results in a conclusion where each involved country per time frame gets a percentage value of how much retweets it's produced against the other countries.

5. Share of retweets regarding to all tweets for each country

program file	`TweetRetweetsShareCountry.java`
property file	`tweetretweetssharecountry.properties`
window size	15 minutes
total runtime	24 hours
date	*7th Oct 2018 1930* to *8th Oct 2018 1930*
output	Console, Elasticsearch, and CSV
stream (ES index):	twitter-retweets-share-country
stream (ES Visualization):	Retweet/Tweet Share per Country, Retweets per Hour per Country
stream (CSV):	`output/tweet-retweets-share.country.stream.csv`
data:	`data/tweet-retweets-share-country.csv`
batch (CSV):	`output/tweet-retweets-share.country.batch.csv`

The last metric calculates the amount of retweets to the over all sum of produced tweets in a given time window per country.

Comparison

The three comparison task in the project can be found in the `comparison` package. This project uses the streaming mechanism of metric 1, 4 and 5 and compares the current online values which its batch generated values over the same daytime. The results of the comparison can be found in the attached video.

Predictions

This project includes three predcition types for predicting the amount of retweets in the next given time frame. These three prediciton types are:

1. Prediction through taking the average over the last 5 amounts of retweets
2. Prediction through a simple regression model
3. Prediciton through a bandit-problem inspired reinforcement learning model for non-stationary reward tracking

program file	`prediction.TweetRetweets.java`
property file	`tweetretweets.properties`
window size	1 minutes
total runtime	6 hours
date	*25th Oct 2018 1700* to *26th Oct 2018 1930*
output	Console, Elasticsearch
stream (ES index):	twitter-retweets-prediction
stream (ES Visualization):	Prediction: Retweets per Time

All of these predcitions get calculated in a stream processing environment and by the calculation of the sum of retweets per time unit.

In each prediction the type of the prediction, the real amount of retweets, the predicted amount of retweets and the deviation are getting saved in certain csv files, passed to elastic search or getting printed in console.

In kibana there is also a visualization which shows the real amount of retweets per time according to the results of the three different predictions.

Processing & Results

The results of the stream processing can be found in elasticsearch under the different indices. When using the given docker environment and the mounted folder `env/esdata-latest` all indices and the corresponding visualizations can be found in Kibana through the URL `localhost:5601`. The pure results are also written in corresponding csv files for comparison purposes with the batch processing. You can found these in the `output/`folder. Beside these two output forms, the pure incoming data gets written into the `data/` folder while the stream job is running. This derived data gets processed by the batch jobs later on.

The results of the batch processing got written into single csv files as well.

The comparsion of batch and stream processing happens in the `de.moritzkanzler.comparsion/` module. Here both output files for batch and stream from the `output/` directory get loaded and processed for comparison.

The prediction part of the project happens in the `de.moritzkanzler.prediction/` module. Here a stream of currently incoming twitter data gets processed and different types of predictions get applied.

Demo video

The demo video can be found as an extra zip under the following link https://www.dropbox.com/s/l3id6gred48hv8q/Big%20Data%20Techniques.mp4.zip?dl=0 or via Youtube under https://youtu.be/vIbxQQyKe_U. The time table underneath allows you to directly jump to different sections:

1. Folder structure (00:18)
2. Programm structure (02:35)
3. Results & metric 1 (05:52)
4. Metric 3 (07:50)
5. Metric 4 (10:16)
6. Metric 5 (11:43)
7. Predictions (12:54)
8. Comparison: Overview (17:28)
9. Comparison: Results (20:38)

References and Resources

- Coordinates of capital cities: http://techslides.com/list-of-countries-and-capitals