University of Pisa

SCUOLA DI INGEGNERIA

Corso di Laurea in Artificial Intelligence and Data Engineering

Data mining project documentation

Candidati

Giacomo Mantovani

Stefano Poleggi

Relatori

Prof. Francesco Marcelloni

Prof. Pietro Ducange

Anno Accademico 2019–2020

Contents

1 Introduction

The **TweetQuake** application offers a real-time earthquake detection service in Italy. When the application opens up it will start fetching tweets and analyzing them to recognize ones related to an earthquake. The application will show the number of tweets analyzed so far and the corresponding number of earthquake tweets and non-earthquake tweets. When some earthquake tweets are recognized the application show a warning message, if there is a trend of earthquake tweets then the application will show an emergency message.

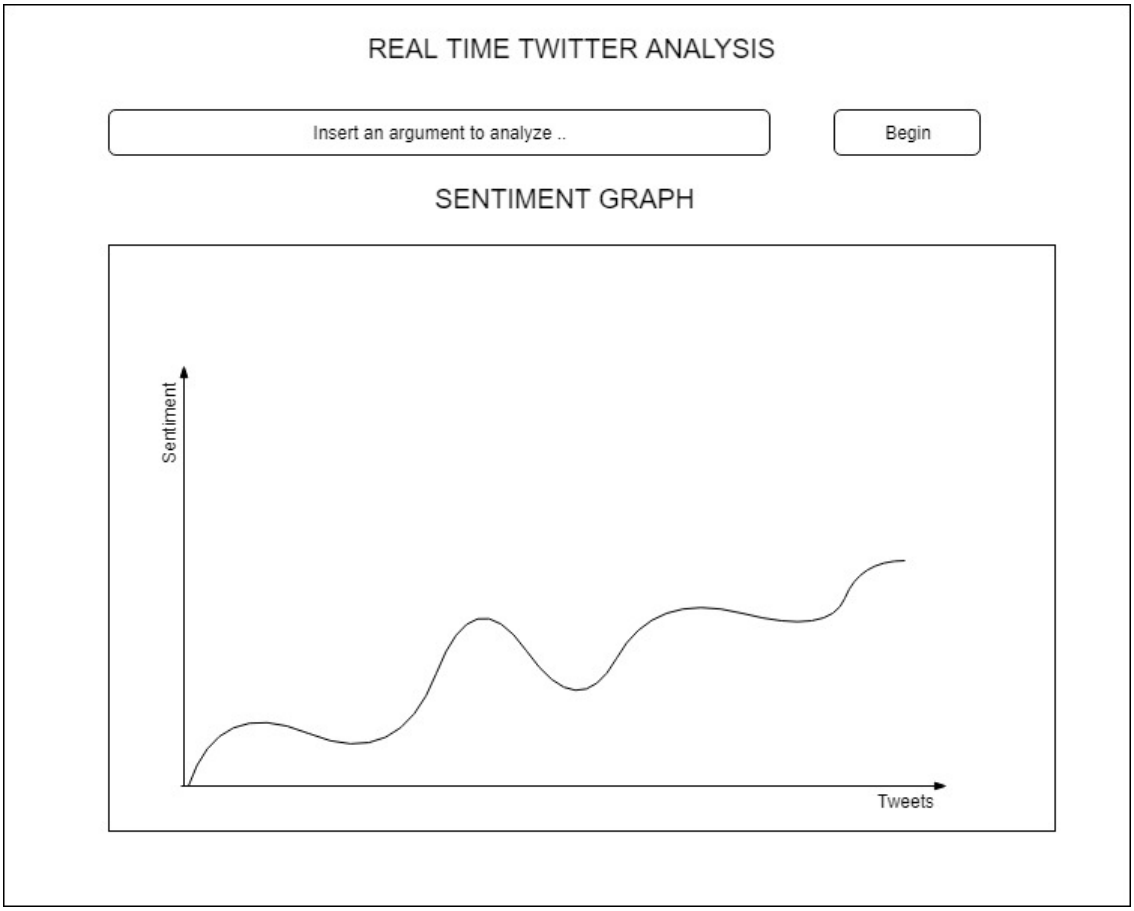

Figure 1: Home Page Mockup

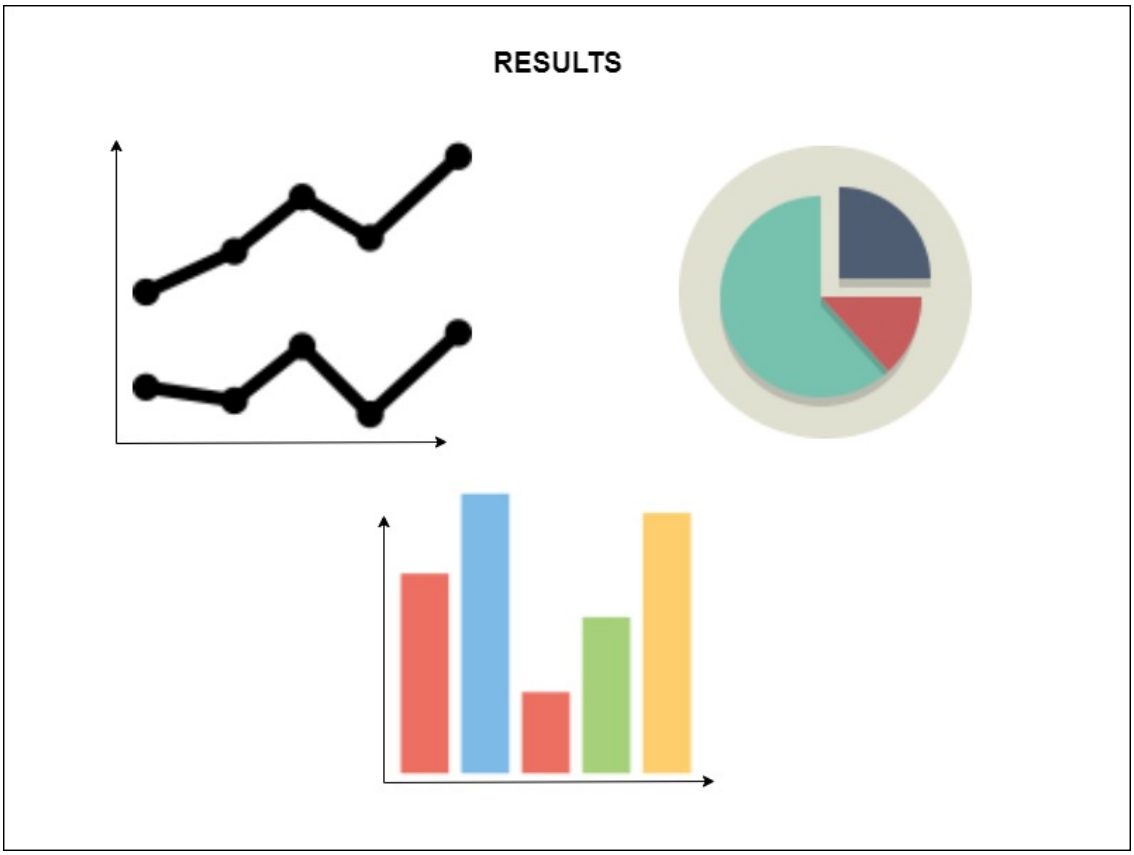

Figure 2: Results Page Mockup

2 Analysis and workflow

2.1 Requirements

2.1.1 Functional requirement

The system has to iteratively perform the following operations:

- Real-time fetching of tweets.
- Perform a classification of the tweets and obtain the label (earthquake or non-earthquake).
- When an earthquake tweet is regocnized send a warning message.
- When an earthquake tweet trend is recognized send an emergency message.

2.1.2 Non-functional requirements

- Usability, ease of use and intuitiveness of the application by the user.
- The system should provide a high level of accuracy.

2.2 Use Cases

Actors

- User: this actor represents a user of the application
- System: this actor represent the system
- Twitter: this actor represent the Twitter service

2.2.1 Use Cases Description

- Twitt Search : This use case can be performed by the user to start the real-time tweet fetching.
- Retrive Tweet: This use case represents the action of getting a tweet from twitter.
- Twitter Connection: This use case represents the connection with the Twitter service.
- HTTP Request for Connection: This use case represents the Request for the connection to Twitter Service.
- HTTP Response for Connection: This use case represents the Response for the connection to Twitter Service.
- HTTP Request for Tweets: This use case represents the Request for tweets.
- HTTP Response for tweets: This use case represents the Response for tweets.
- Perform Text Analysis: This use case represents the process of Text analysis performed by the system (for more informations check paragraph < add paragraph num >)
- Apply Classification Algorithm: This use case represents the classification of the tweets performed with the classifier generated by the "Generate Classifier" use case.
- Show Warning/Emergency Message: This use case displays an error message if some earthquake tweet are recognized.
- Display Results: This use case shows the results of the classification.
- Generate Classifier: This use case is performed automatically by the system and it generates a classifier using a training set.
- Browse Tweets: The system browses the data on the database.
- Train the classifier: Perform an algorithm to generate the classifier.
- Stop Real-Time fetching: This use case allows the user to stop the real-time tweets fetching.

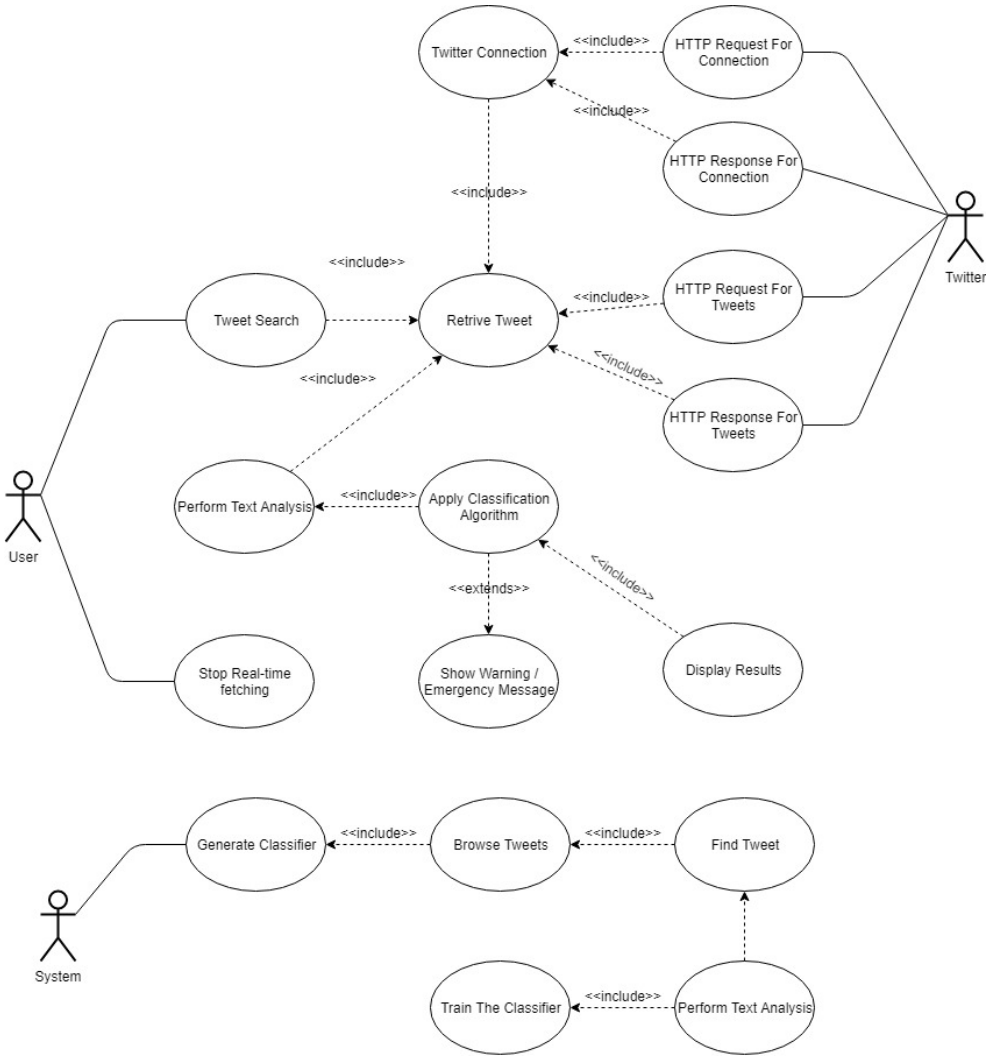

Figure 3: Use cases diagram

2.3 System Flow Diagram

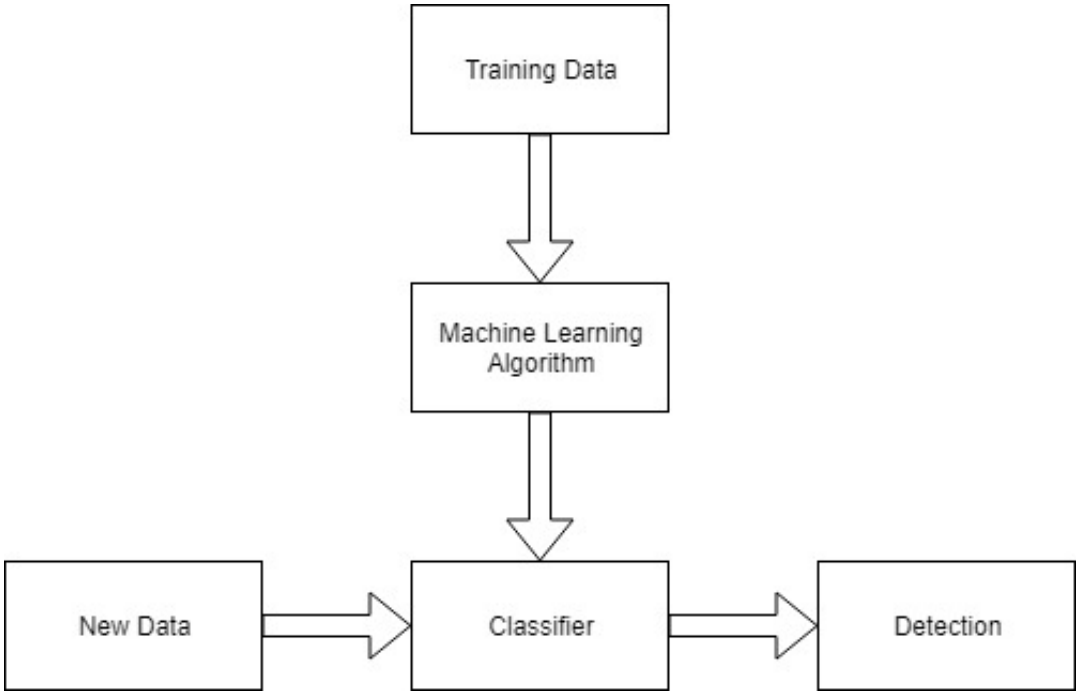

Figure 4: Flow Diagram

2.4 System Architecture

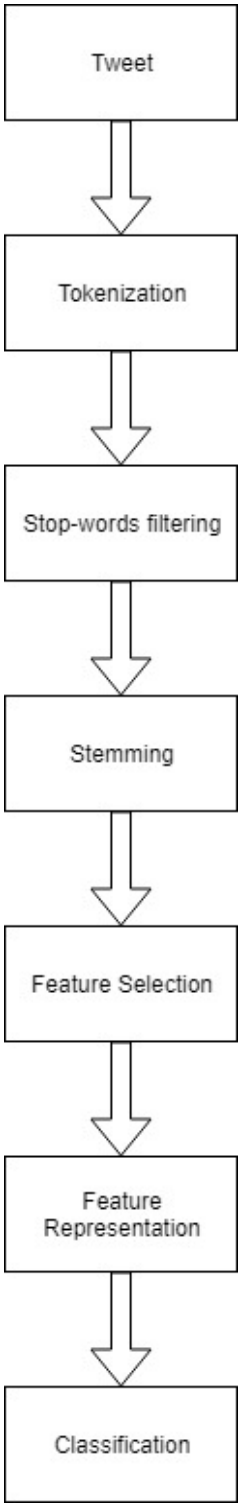

Figure 5: Text Analysis Process

2.5 Analysis of entities

This diagram represents the main entities of the application and the relations between them.

Figure 6: UML analysis diagram

3 Design

3.1 Database Choice

After the analysis phase, which has been carried out so far, we start with the design of the **TweetQuake** application. We decide to use MongoDB as data support. Its document-based structure is very useful for the large amount of data that we need to maintain and access, as well as its high scalability, qualities that we do not find in a relational database.

3.2 Software architecture

The application is designed over 3 different layers, see figure 7:

- Front-end
- Middleware
- Back-end

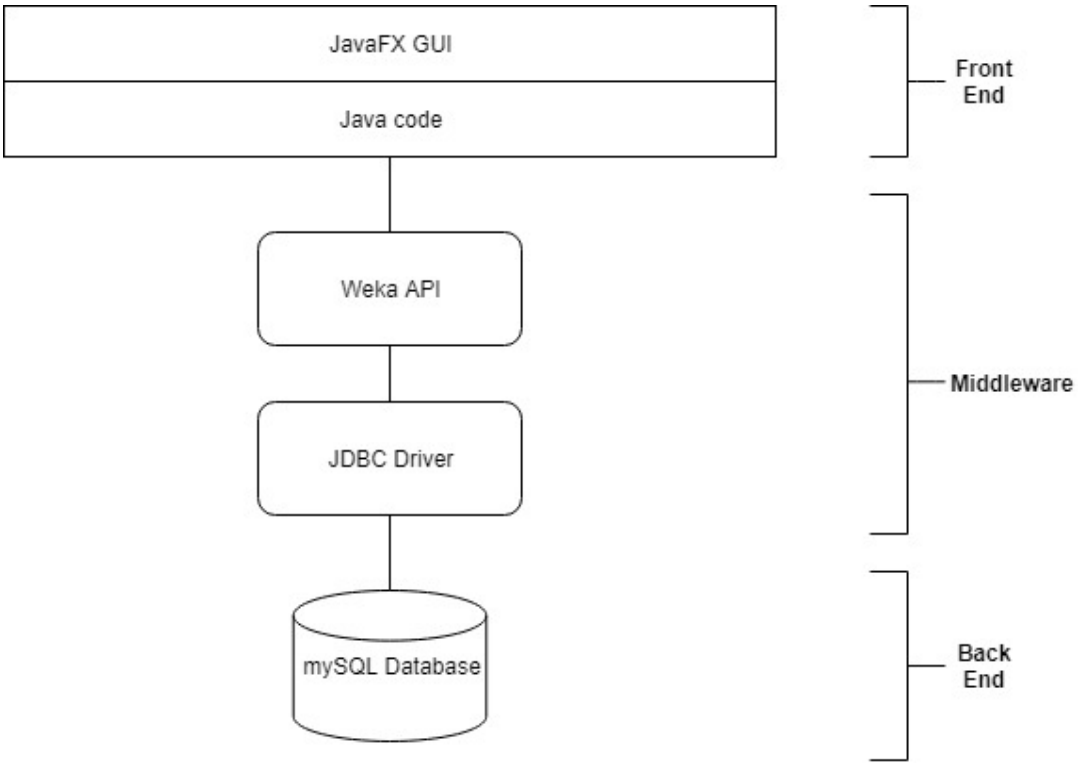

Figure 7: Software architecture diagram

3.3 Populating the database

The dataset used in the **TweetQuake** application was created by scraping Tweets, using Twitter4j which is a java library for the Twitter API. Given location and tags to search for, this API returns the informations about the tweets matching those parameters. To build the Database two different queries have been used:

- The first query selects the tweets in Italy matching the tags #terremoto, #magnitudo and #earthquake.
- The second query selects the tweets in Italy without tags.

After that the text of the tweets have been stored in the database and the labels have been added manually to obtain a good training (and test) set to generate a classifier. The second query is necessary to obtain better results because without the tweets obtained from it the classifier would suffer from overfitting.

<code here>

4 Implementation

4.1 Used technologies

The application is developed in java programming language, version 11.0.4, and in JavaFX system to create the GUI, version 11, so it should run on each platform in which JVM is installed, but the application is tested and guardantee on Ubuntu 16 and Window OS. Moreover Maven is used to build and mantain the project, version 3.8.0.
The java driver for mongo manage the comunication between client application layer and mongo backend layer, version 3.11.2.
For the backend layer it is used MongoDB, version 4.2.
So this application is tested using these technologies, considering these particular versions: for other versions the correct execution isn't guaranteed .

4.1.1 Declaration of class Tweet

The following java-code shows a declaration of the class Tweet.

<code here>

Another foundamental class is SQLManager, that manages the db-connection and the related operations.

4.2 Create

Adding a tweet to the database.

<Code here>

4.3 Read

This functionality returns a list of tweets.

<code here>

4.4 Text Analysis

This functionality performs a text analysis of a tweet.

<code here>

4.5 GUI

There is an fxml documents which describes the objects showed in the GUI interface of the page.

- Home.fxml

In addiction there is a class, called Controller, that is in charge of handling events of the objects defined in the associated fxml document.

- HomeController.java

5 User Manual

When you first run the application, the interface you get is the login one, figure **??**.

<login page image–> image name: screen0>

In case you are not registered you can click the link at the bottom of the page to be redirected to a register page, figure **??**, otherwise you can sign in the application and by default you get the interface shown in figure **??**

<register page image –> image name: screen1>

<Application home page after login –> image name: screen2>

From here you can search for a film by typing in the relative field and clicking the search button. You'll get a list of films that contains the text entered in the table below. Now you can select a film from the table and all the informations will be shown in the right pane of the application, figure **??**. In the bottom right you are able to add a vote from 1 to 10 for the selected film.

<homepage after searching a film and selectiong one –> image name: screen3>

In the top left of the page there are two tabs. by default after the login you are in the Home tab, by clicking the Analytics Tab you will see the following page.
<analytics tab page –> image name: screen4>

Here you have a choice box (left) to select what kind of analytics you want to be performed from the existing ones, and another one to apply a filter (right), figure **??**.

<default analytics page –> image name: screen5>

After selecting an analytics you will see the results in the table below, and for some of them you will get a piechart aswell, figure **??**

<analytics page after performing analytics –>image name: screen6>

To log out, just click on the appropriate button at the bottom.