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. If the button at the end of the page is clicked then a new window containing the last recognized earthquakes opens up.

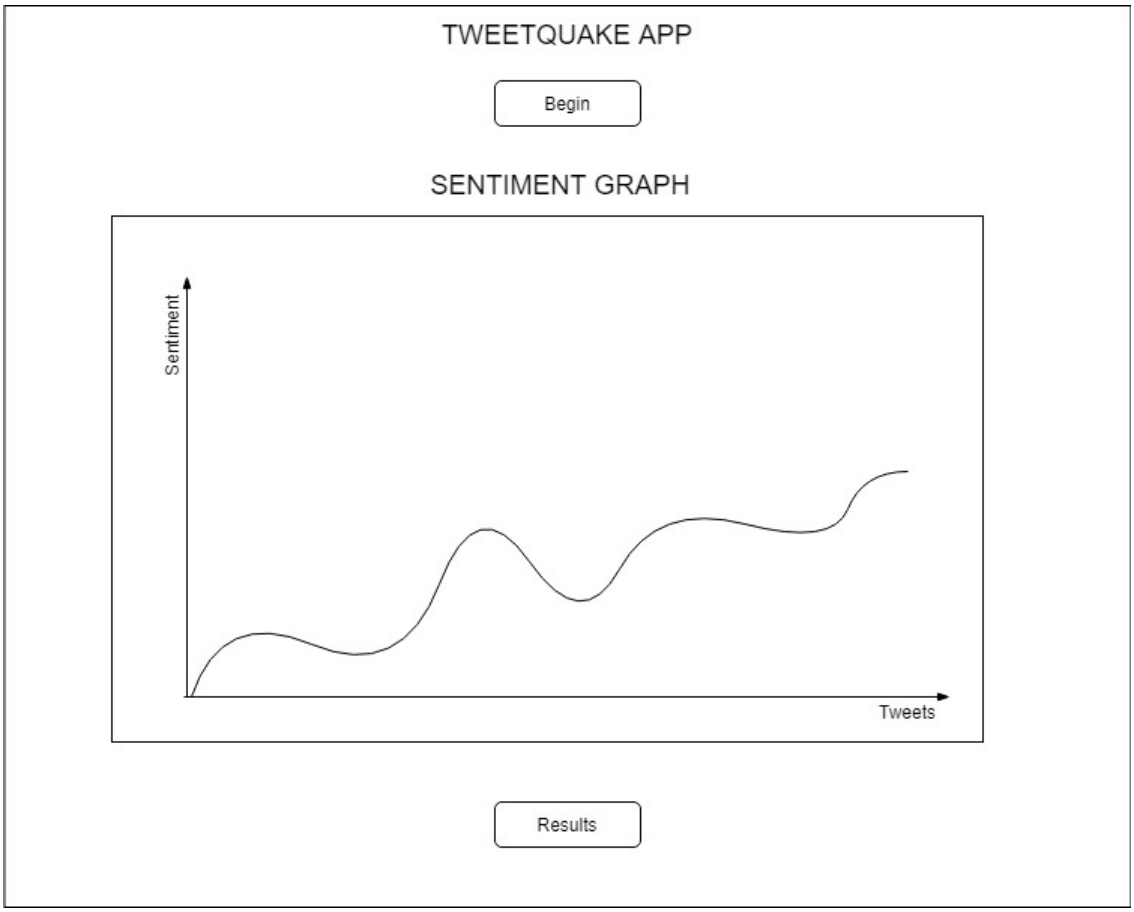

Figure 1: Home Page Mockup

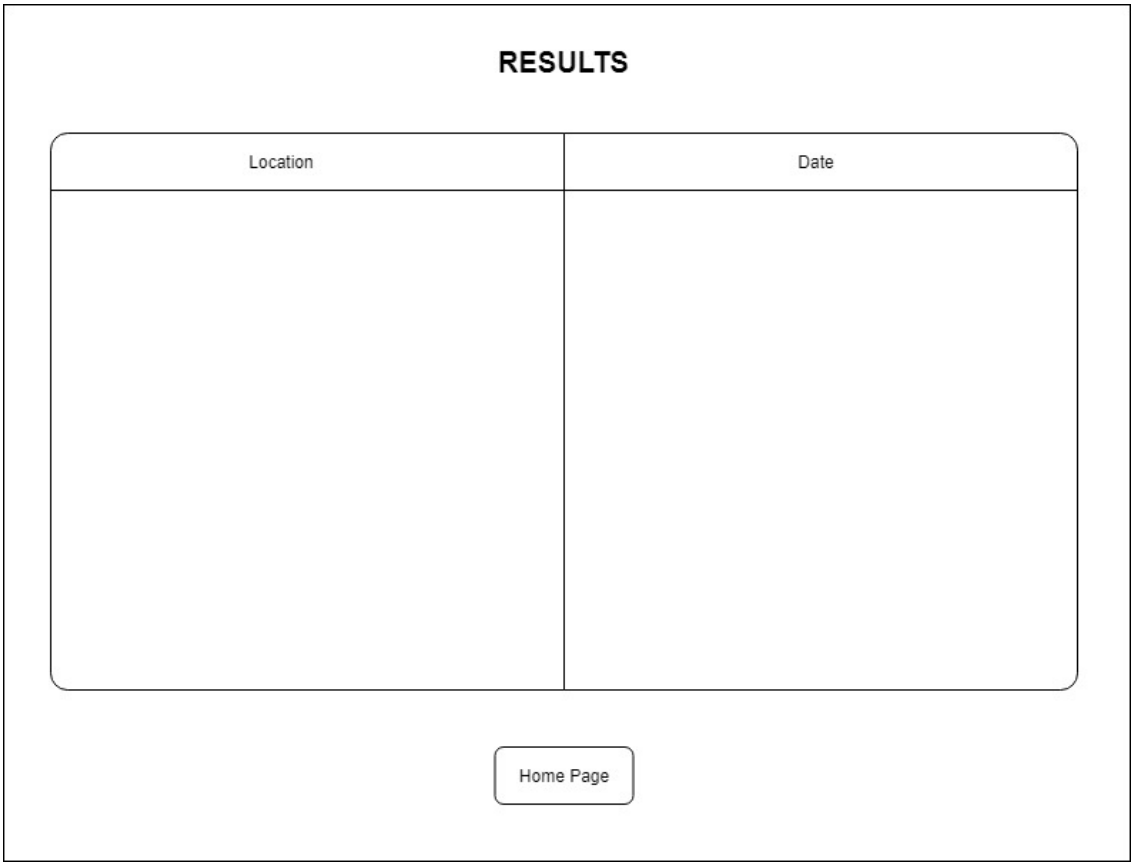

Figure 2: Results Page Mockup

2 Analysis and workflow

2.1 Requirements

2.1.1 Functional requirement

The system has to allow the user to carry out basic functions such as:

- To start the text analysis.
- To stop the text analysis.
- To view the most recent recognized earthquakes from the application.

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 and add the relative earthquake information into the database.

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 tweets 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 the Twitter Service.
- HTTP Response for Connection: This use case represents the Response for the connection from the Twitter Service.
- HTTP Request for Tweets: This use case represents the Request for tweets to the Twitter service.
- HTTP Response for tweets: This use case represents the Response for tweets from the Twitter service.
- Perform Text Analysis: This use case represents the process of Text analysis performed by the system.
- Apply Classification Algorithm: This use case represents the classification of the tweets performed with the previously generated classifier by the "Generate Classifier" use case.
- Show Warning Message: This use case displays an warning message if a few earthquake tweet are recognized.
- Show Emergency Message: This use case displays an emergency message if a some earthquake tweet are recognized.
- Add Earthquake To Database: This use case represent the insert of the recognized earthquake into the database.
- Display Results: This use case shows the results of the classification.
- Browse earthquakes: The system browses the earthquakes on the database.
- Find earthquake: The system selects an earthquake on the database.
- Display Earthquake: This use case shows the earthquake on the application.
- 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 tweets on the trainig dataset.
- 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.

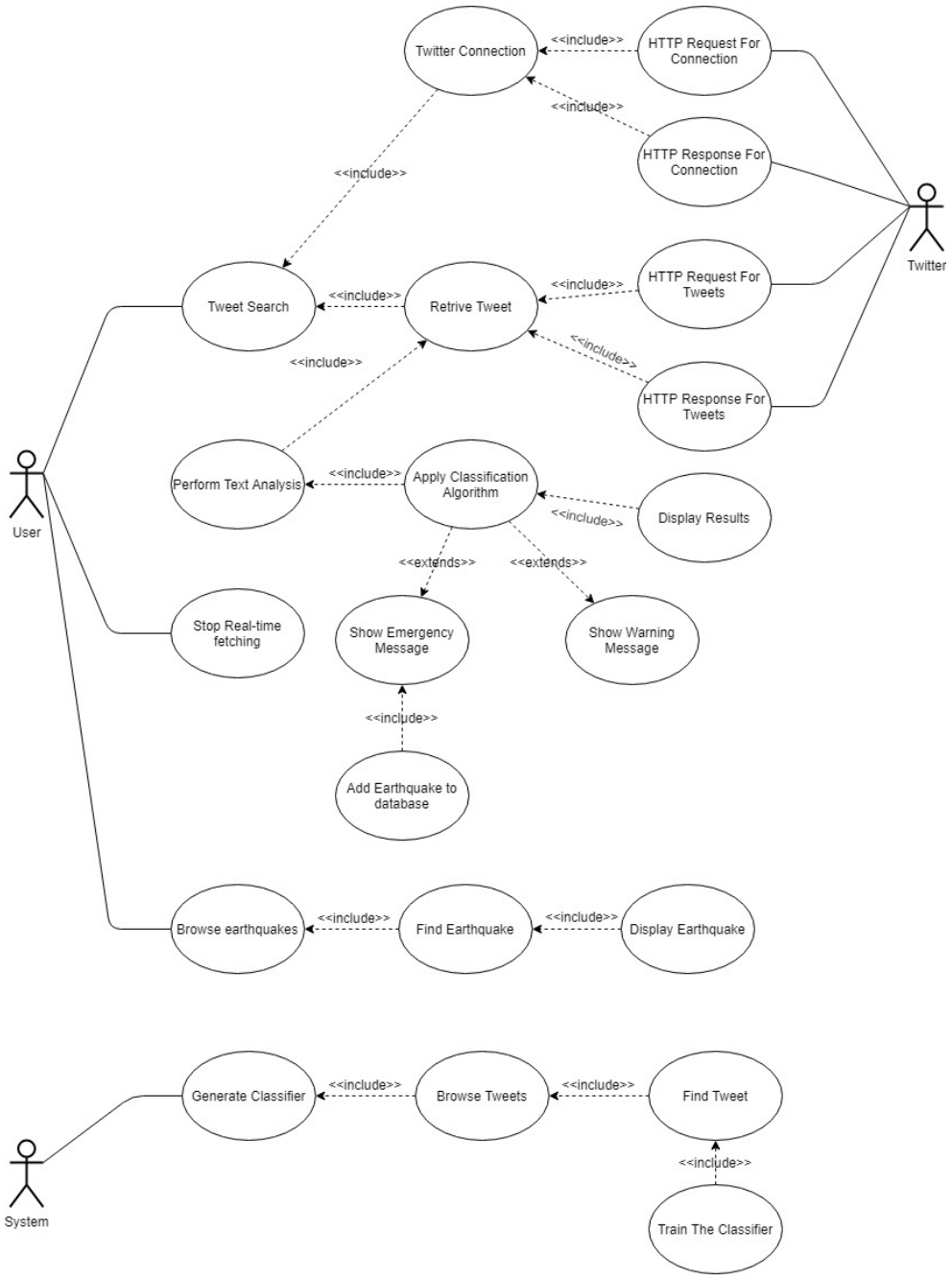

Figure 3: Use cases diagram

2.3 System Flow Diagram

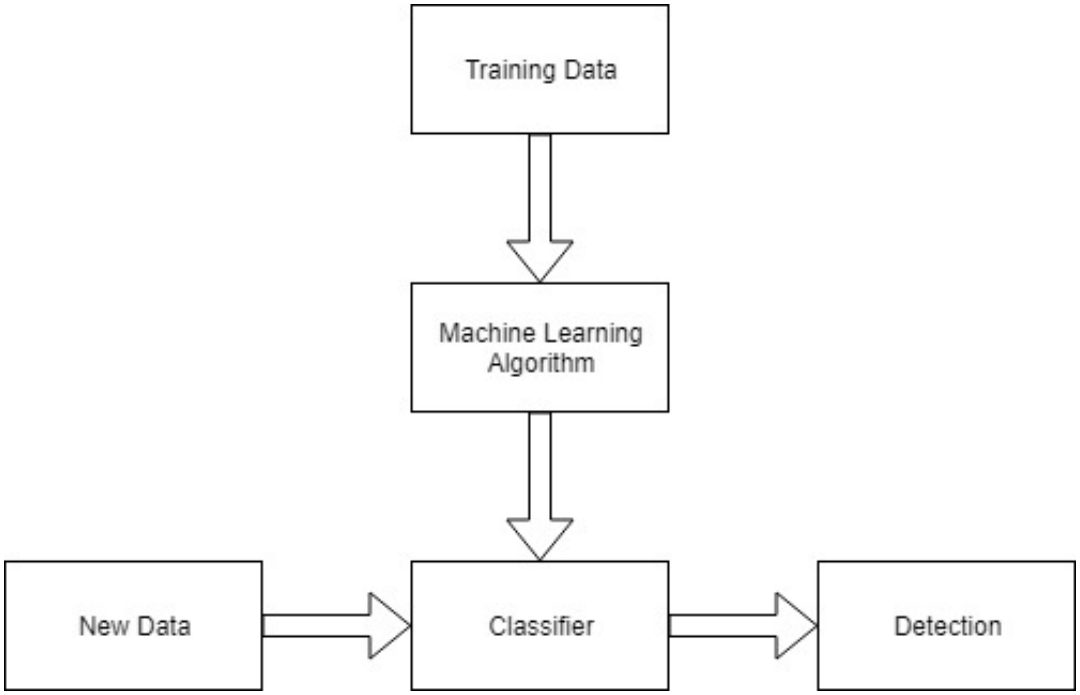

Figure 4: Flow Diagram

2.4 Text analysis process

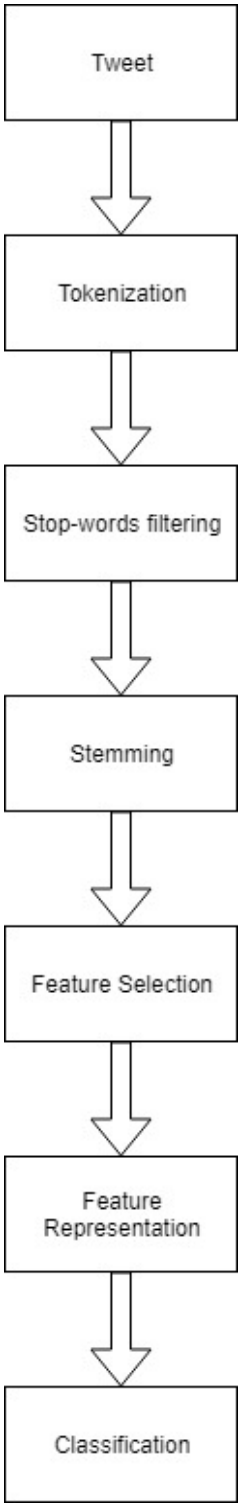

Figure 5: Text Analysis Process

2.5 Analysis of entities

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

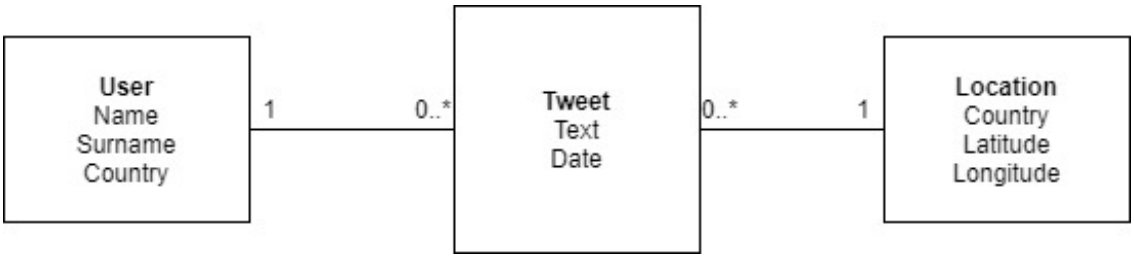

Figure 6: UML analysis diagram

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 mySQL as data support since we don't have to store large amount of data in the database and we don't need a high-performance database. In the database we will store only the informations about the recognized earthquakes. The training tweets will be temporarily stored into an SQL database and after some operation the training dataset will be saved in an arff file (and no longer into the SQL database).

3.2 Software architecture

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

- Front-end
- Middleware
- Back-end

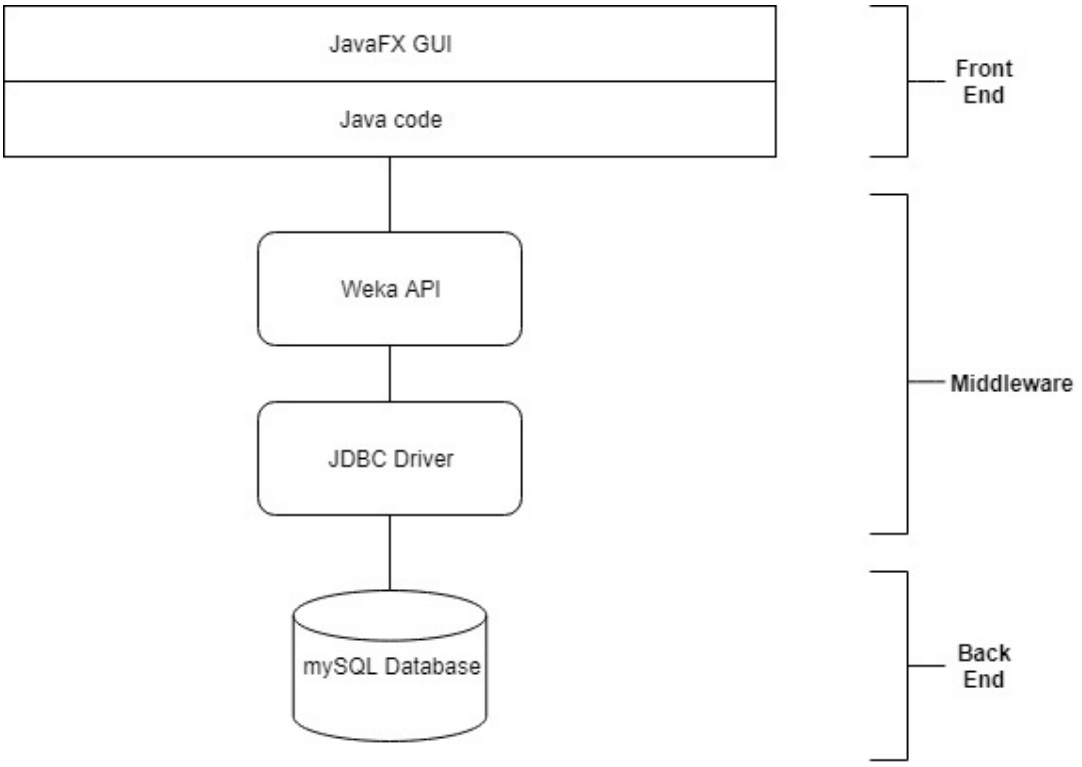

Figure 7: Software architecture diagram

3.3 Building the classifier

In order to build our classifier raw tweets have been scraped 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 temporarely save those tweets a mySQL database is used. To build the Database two different queries have been used:

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

After that the labels (earthquake, non-earthquake) have been added manually to obtain a good training (and test) set to generate a classifier. We ended up with a balanced dataset of 700 tweets per class, 1400 labeled tweets.
The training data will be stored in an arff file, after cleaning the text of the tweet and applying tokenization, stop-words filtering and stemming. The second query is necessary to obtain better results because without the tweets obtained from it the classifier would suffer from overfitting.
Once the training data is ready a classification algorithm is applied in order to build the classifier.

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 background layer it is used Apache as web-server, version 2.4 (including MySql).
So this application is tested using these technologies, considering these particular versions: for other versions the correct execution isn't guaranteed .

4.2 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.

4.3 Building the classifier using RandomForest

The following java-code shows the training part of the process to obtain our classifier using the random forest classification method.

<code here>

The choice of the random forest method has been made by testing multiple classification algorithms such as NaiveBayes, ... and compare them. Since the random forest seems the most accurate from the tested ones it has been choosen for the TweetQuake application.

<table comparing classification models>

The confusion matrix obtained from the classification is the following:

<confusion matrix of the classifier>

4.4 Create

Adding a tweet to the database.

<Code here>

4.5 Read

This functionality returns a list of tweets.

<code here>

4.6 Text Analysis

This functionality performs a text analysis of a tweet.

<code here>

4.7 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