Data Intensive Computing
Final Project Report

Luca Colasanti, Davide Marinaro

24 October 2021

1 Problem description

Weather forecast is typically a very challenging task because there are many variables to consider and the availability of measurements is often overwhelming and very difficult to deal with efficiently. Thus ensuring a consistent and fast way to deal with streams of measurements is key in achieving good performances. The main prerequisites for the success of this project are an adequate architecture dealing with data retrieval, a secure way to store data persistently and finally an efficient solution to perform predictions on real time samples for actual weather forecast in a reasonable timescope.
The type of data required is abundantly available by querying for example the average weather conditions of a city. Generally weather represents a very **diverse and rapidly changing aspect** to predict since it easily changes during the day and depends on the geographical location of the queried city. For these same reasons it allows for a colorful collection of samples on which to build a forecasting model.The aim of our project is to utilise a sort of three-tiered architecture (data, application and visualization) to retrieve live data from a weather API, perform predictions on it and finally provide an explanatory visualization of it. In our application, data will be mostly coming from cities since weather stations are not evenly spread across all the territory. We have chosen Italy as the source of data and consequently we will make predictions on the Italian territory. We have chosen this use-case because we think it is an interesting challenge to practically understand the mechanics of the **distributed computation**, dealing with **streaming data**, **machine learning** on a **distributed dataset**, and **visualization** of data stored on a distributed environment.
The goal of the project is to answer to the following questions:

- What will be the temperature and the weatherforecast of a given Italian city 3 hours from now?

2 Data

To fulfill the goals of the project we need both **historical data** and **real-time data**, which will be gathered using the OpenWeatherMap API [1].

We built a historical dataset gathering measurements from 20 cities once a hour for several days, collecting an overall amount of around 7000 records. These measurements are composed of the following data:

- **Name** of the city (String).
- **Timestamp** of the measurement.
- **Latitude** and **longitude** of the city.
- Measured **temperature** (in Kelvin).
- **Felt temperature** (in Kelvin).
- **Pressure** (in mBar).
- **Humidity** (in percentage).
- **Label** for the weather, which can either be:
  - *Clear*.
  - *Clouds*.
  - *Fog*.
  - *Rain*.
  - *Snow*.

These measurements are statically used to train the **ML models**, a regressor and classifier, that are then used to predict on the streaming data retrieved from the API to perform weather and temperature forecast.
The records that are obtained in real time to perform prediction on are of the same structure as the ones used in the historical dataset, with particular relevance to the geographical parameters used to locate the city in the visualized map. The final structure of the record the consumer sends to the visualization function is comprised of the same columns as before plus:

- **Forecast temperature** measured in Kelvin (Displayed in Celsius [°C]).
- **Label** for the forecast weather.

3 Tools

The tools we used in the project are:

- Main Language: **Scala**
- For the stream processing: **Kafka**

- For data preprocessing: **Python**
- For the distributed storage: **Cassandra**
- For the classification and regression models: **MLlib**
- For the visualization: **D3.js, Javascript** and **HTML**
- For the web platform: **Node.js, Javascript** and **HTML**

4 Methodologies

The first task to perform has been to build the historical data set by querying the **OpenWeatherMap API** over the course of several days, gathering hourly records from a list of 20 Italian cities, like Milan, Rome and Naples just to mention the most relevant ones, The records have then been manipulated by reducing the redundant labels used to indicate the weather forecast to only five. This data was later to be used for the training of the ML models. Regarding the ML models we have decided to design a couple of them:

- A **Gradient-Boosted Tree** to perform regression on the forecast temperature in 3 hours based on the other parameters already listed in the description of the data structure.
- A **Decision Tree** to perform classification on the new datapoints obtained by appending the predicted temperature to the dataset, in order to infer the label that indicates the weather forecast in 3 hours.

The **streaming pipeline** instead starts from **Kafka** messaging services. A **Producer** is responsible for periodically (every 1 second) retrieving a batch of 20 record from OpenWeatherMap API on which to perform prediction through the **regressor** and **classifier**. Data gets sent to the topic "Weather".
A **Consumer** is instead responsible for reading the data, checking for any errors and conforming the labels to the subset we defined, deleting redundant ones and then feeding said labels to the **ML models**, all the while sending them to the topic "Forecast". The Consumer then exploits **Cassandra** to save the new parsed records. This step could have been avoided because we do not exploit the functionalities of **Cassandra**, but we decided to implement the data storing in our pipeline in order to have the possibility of updating the **ML models** on richer datasets in another moment later on. The **Consumer** then forwards the records to the visualization interface, where geographical and weather attributes are interpreted to visualize the record and correctly locate on the miniature Italian map. In order to avoid overly burdening the system, records visualizations are removed from the map after an offset of 90 seconds.

5 How to run the code

Requirements:

- Java 8 SDK with sbt
- Python 3
- Scala
- Apache Spark
- Apache Kafka
- Apache Cassandra
- D3.js
- Node.js v14.18.1
- npm v6.14.15

The code was tested on Ubuntu 20.04 LTS, with the following environment variables:

```
export KAFKA\_HOME="/path/to/the/kafka/folder"

export CASSANDRA\_HOME="/path/to/the/cassandra/folder"
```

Finally, there are several ways to run the code. The simplest one is to enter the folder in which the file `start-server.sh` is located, open it in terminal and run the following command: `./start-server.sh` Then we simply have to visit `http://localhost:14520` to observe the results. There is another file with a structure similar to `start-server.sh`, named `start-script.sh`, which includes the automatic opening of the web page, but it is not recommended to use. The second method to run the code is to manually type the following commands in the ubuntu terminal:

```
zookeeper-server-start.sh $KAFKA_HOME/config/zookeeper.properties

kafka-server-start.sh $KAFKA_HOME/config/server.properties

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic weather

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1
--partitions 1 --topic forecast

cassandra -f
```

Enter the `producer` folder and type the following command:

```
sbt run
```

Enter the `sparkstreaming` folder and type the following command:

```
sbt run
```

Enter the `app` folder and type the following commands in sequence:

```
npm install
npm start
```

After the servers are active, visit `http://localhost:14520` to see the outcome.

As an additional material, alongside this report, a sample recording showing the behavior can be found at the official GitHub page of the project or in the `FinalDeliverables` folder.

6 Results

The results of the weather forecast are shown through a colour legend on a map representing Italy, to which all records are mapped given their latitude and longitude.

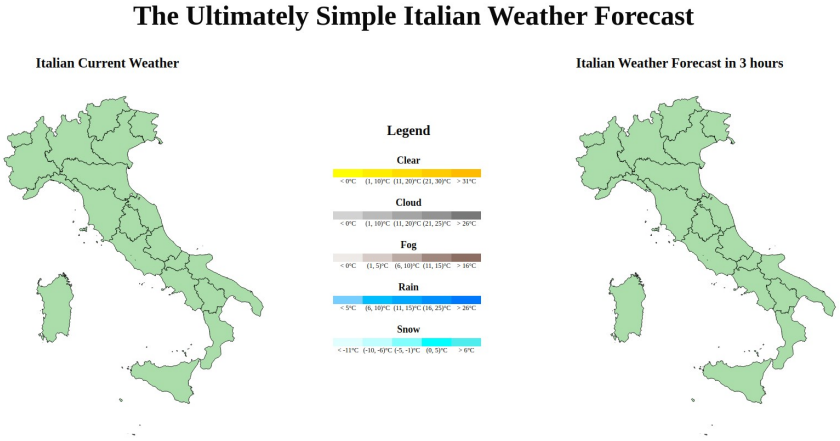

Figure 1: Example of the visualization map on which records are showcased.

Now instead we provide a visualization of the end results.

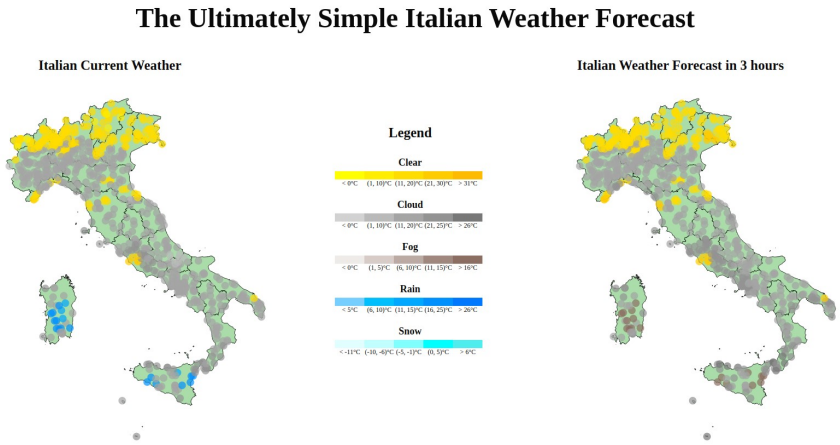

Figure 2: Example of weather current conditions and forecast after 3 hours performed on Sunday October 24th 2021 at 5:45pm.

As one can observe from the visualization in 2, we decided to represent each weather record as a circle on the map that has a representation of the weather via colour scheming (Yellow: Clear, Grey: Cloud, Grey: Fog, Blue: Rain, Light blue: snow), while the intensity of the colour indicates the temperature of the given location. The same stands for the predicted weather forecast and temperatures, which are visualized on the right hand side and follows the same coding.

It is worth to highlight that the ML model behaves with an **accuracy** of 75.6% for the **classifier** and **RMSE** of 1.23 for the **regressor**, given the decently sized dataset.

Our focus has not been to provide extremely reliable and on point forecasts, rather showcase the capability of **implementing a functional and lightweight ML functionality** on top of an efficient architecture capable of handling streaming data. Further expanding the datasets and refining the prediction functions the results are sure to improve the overall performance.

In general the model performs quite well and manages to keep the outputs coming consistently, making the whole **architecture feasible** and for sure **expandable** to perform further manipulation and prediction on the data. It is worth mentioning that as of now we limited ourselves to highlight the **distributed efforts** of the whole architecture, but the usage of persistent storage can be further optimized and integrated for other kinds of analysis with the build-up of an overarching application capable of providing further insights, as well as an even more appealing dashboard for visualization.

7 References

[1] https://openweathermap.org/