Forecasting imbalance risks for Italian power grids using renewable energy predictions

Gabriele Ghisleni Alessandra Pomella

University of Trento
Email: gabriele.ghisleni@studenti.unitn.it, alessandra.pomella@studenti.unitn.it

The present system was developed with two main goals. First, to predict the amount of energy that can be produced in Italy from renewable sources (hourly on a daily basis). Secondly, to estimate the hourly imbalance risk that might affect the Italian power grid in its performance of duties. We combined weather and energy generation data to train predictive models, then we exploited those algorithms for inferring power capacity and possible risks of excessive or insufficient electric supply. The results are delivered with a web app providing tabular and graphical representations of our predictions up to 48 hours in the future.

Keywords: Renewable Energies; Load; Green Energy; Imbalance; Forecasting; Terna; MySQL; Redis; MQTT; Queueing Systems; Pub/Sub; Docker; Docker-compose; Amazon Services; Statistics; Analytical Models; Encapsulation; OOP; Django; Web App; Big Data System;

1. INTRODUCTION

Before discussing the system architecture together with its technological aspects, we briefly focus on some preliminary issues that were discretionary and crucial to be pondered. Those preliminary decisions (ranging from the concept of power imbalance to the choices of both data and predictive models) defined the coordinates of our system design and oriented its implementation.

1.1. The Concept of Power Imbalance

First of all, given that the final goal of the project was to estimate power imbalances, it was crucial to determine which deviation among energy supply and energy demand is technically considered an *imbalance* in the electricity field. And, moreover, which are the elements to be monitored that, in particular combinations, usually generate this phenomenon.

After some research [1], we decided to define the potential imbalance concept as follows: we say that a power imbalance might occur when the absolute distance between the total load and the total actual generation (as a sum of the predicted energy production from both renewable and traditional sources) results to be greater than zero. The above mentioned elements to be considered are then the *total load* (general amount of required energy) and the *energy production* (photovoltaic, geothermal, wind, biomass, hydro and thermal respective predicted quantities of power generation).

1.2. Data Description

For model training purpose, we needed both an historical set of weather data and a concurrent dataset about the actual Italian power generation capacity divided per source, every hour and over the days of an entire year. Given that no free weather historical database was actually accessible, we were forced to auto collect those data through the OpenWeather API exposed by the OpenWeatherMap website [2]. This operation was repeated every hour for days, taking advantage of an EC2 t.2 micro instance of Amazon Web Services that, being always up and running, allowed us to gradually populate a DB. Thus, this set of operations resulted in a MySQL DB (via Amazon RDS) available on AWS, containing observations ranging between April 2021 and July 2021.

Power generation data were instead collected from Terna S.p.a. download center. We chose Terna as the appropriate data source due to the fact that the company, being the first grid operator for electricity transmission in the Italian power system, collects our target power data constantly. We manually downloaded their datasets about load, renewables and traditional thermal power from the website (data with hourly rate and a national aggregation level). Given that the energy data were not available on a regional accuracy degree, we aggregated the Weather observations pertaining the different Italian regions so to have one single vector of observations per hour instead of twenty.

All the data we were brought to use were structured in tabular form, so relational databases were the obvious

choice. Lastly, all data were batch processed: the prediction rate we set would not have required nor justified an attempt of stream processing.

1.3. Model Assessment and Selection

The intermittent, hard to predict and even harder to store nature of the renewables (especially when it comes to photovoltaic and wind) affects the whole power delivery system. The issues regarding the nature and the trends that differently characterize each energy source are connected to the way we built the respective predictive algorithms, testing and selecting the best model for each power source.

For *photovoltaic*, *wind* and *biomass* power we privileged random forests models (in one case using in particular bagging regressors) for performance reasons, resulting from cross validation tests.

For *geothermal* estimation we used a dummy regressor adopting the historical mean as predictor, due to the fact that it is evidently constant and widely considered a "renewable, sustainable form of energy that provides a continuous, uninterrupted supply of heat that can be used to produce energy"[3].

For *thermal* forecasts the approach was different. And the same thing happened for *hydro* predictions when we noticed how similar those trends were to each other (and to the load).

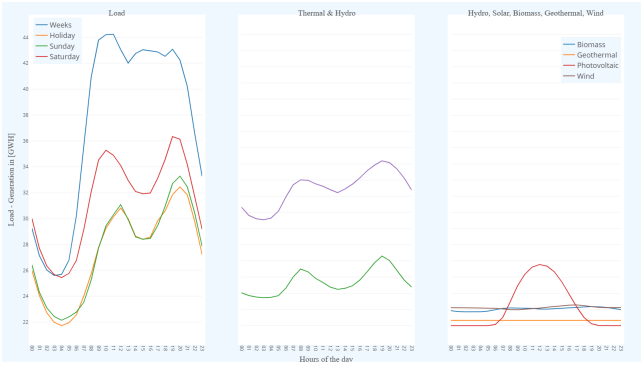

FIGURE 1. Trends for different energy sources

For both *thermal* and *hydro* we opted for a predictive model based on the sum of renewable generation of different sources and on the load at a specific datetime. As a matter of fact, the most steady and reliable renewables, as well as traditional energies, are actually used to compensate imbalances in cases of insufficient supply from other sources. Since hydropower is so reliable, hydro plants can adjust the energy production [4]. A similar situation happens with thermal implants that "provide improved energy efficiency, help secure the power supply and allow a very rapid response to peaks in electricity consumption" [5]. It is then reasonable to use the generation from those other sources together with the estimated load as predictors.

For *load* forecasts we trained the model using historical load data having as predictors the datetime as well as a factor variable called "holiday" (fig.1, left frame): the energy demand fluctuates mainly due to this factor, namely whether it is a working day or not.

2. SYSTEM INFRASTRUCTURE

We built our application following a "*What if the data we're dealing with can be described as Big Data?*" leitmotiv. Hence, we took care of the 5 V's aspects and tried to implement a system that could handle those kind of situations.

In particular, we stress the importance of decoupling processes. We aimed to divide the whole process into small, isolated parts that communicate exclusively with exposed data through the usage of a pub/sub system. In particular, we focused on writing code intending to create components that do not heavily depend on each other. We tried to keep our system as modular as possible in order to preserve the possibility to safely modify some parts without affecting others and to have a whole structure that is less likely to break down when changes occur.

2.1. Message Queuing Telemetry Transport

We begin with the description of one of the most questionable choices of our project, the usage of a pub/sub system as MQTT. The Message Queuing Telemetry Transport (MQTT) is a lightweight, publish-subscribe network protocol that transports messages between devices. This structure can be used to achieve one of the most critical conditions of a big data system, namely the optimal level of decoupling for a given application. The main reason why we adopted such pub/sub technology is that it allows isolating parts of the application. Each of the sub-processes performs a little and solid step of the pipeline and then feeds its results as inputs to the next ones, forming a long chain of processes.

2.2. Encapsulation and OOP

Moreover, we followed a programming style that tries to ensure Encapsulation, another crucial concept to achieve as much decoupling as possible. The idea of Encapsulation refers to the practice of primarily using an Object-Oriented Programming (OOP) style, which implies a principal use of classes (objects) that allow internal details to be hidden from one class to another. This approach provides a strictly defined interface for different classes for mutual communication, to prevent direct access to the attributes yet still using predefined methods and functions.

2.3. Databases Management Systems

For data storage, we used two distinct technologies having different aims.

2.3.1. MySQL Storage

In RDBMS (Relational Database Management System), data are organized in tables with rows and columns defining relations among them. Information can be handled through the usage of SQL queries (Structured Query Language). This class of databases requires the so-called schema on write, meaning that the attributes of a table must be specified when the table is defined and that they are not flexible. Mostly, the schema of a database is the skeleton structure that represents the logical view of the entire database and it determines how the data are organized and how relations are specified.

We decided to use a MySQL DB as the core of our storage system: the target data for this analysis fit well inside tables. In particular, we deal with two sources of data:

1. Renewables generation having only three attributes: *date, energy, generation*. The load table is very similar, but with the additional argument *holiday*.

TABLE 1 Energy generation and demand

idrow	date	energy	generation
1	2021-01-01 00:00:00	hydro	5.192

2. Weather data representing the Italian weather conditions averaged on the twenty Italian chief towns. **This is a transposed representation of the table!**

TABLE 2 Transpose of the Weather table

Column names	Row values
idrow	1
date	2020-01-01 00:00
clouds	33.6
pressure	1013
humidity	49.5
rain_1h	0.4
snow_1h	0
wind_deg	192
wind_speed	4.9
temp	297

2.3.2. Redis Cache

Redis is a NoSQL DB system that is based on an in-memory key-value data structure store, whose primary advantage is the speed in saving and fetching data.

We used Redis DB mainly to show the results in our final application and in some intermediate steps. Redis helped us speed up the whole process, in particular the graphical parts on the Web application where we needed to retrieve the most up-to-date predictions to display.

Moreover, since we used it for retrieving data in a fixed range of dates (today and tomorrow) we also were able to set the expiration deadline for the previous day's predictions, in order to keep in memory only the needed information while having persisting storage in MySQL DB.

2.4. Docker

Docker is a set of "platform as a service" (PaaS) products that uses OS-level virtualization to deliver software through specific packages called *containers*. These containers are isolated from one another and bundle their own software, libraries and configuration files so that they can communicate and interact only through well-defined channels.

This implies a very easy way to deploy and transfer the application since it arrives with all the functionalities, packages and software (needed to properly run it) already set in place. As we said, the usage of MQTT allowed us to decouple our application into several services that, in theory, could have been dislocated into different machines. Thanks to Docker, we were able to faithfully reproduce the advantages of distributing processes even if the app was running on a single device, given that each container could potentially represent a different machine.

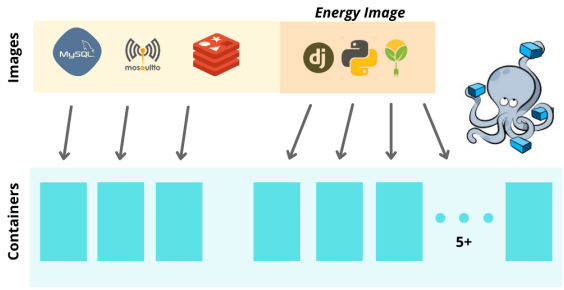

FIGURE 2. Custom Docker APP

2.5. Django

We also decided to develop a simple web application in order to have an endpoint where to display predictions and others functionalities such as a REST API and an overall infographic about the collected data.

This application was built in Django, a high-level Python web framework that enables rapid development of secure and maintainable websites thanks to model template views architectural pattern.

2.5.1. Info Graphics

One main function of the application is rendering the trends of the renewable energies and of the load for the next 48 hours. This is achieved with two plots depicting the load and adding renewable energies trends

stacked one upon each other (see fig. 4). Instead, what's highlighted above the plot is the predicted imbalance, namely the amount of energy required to gap the load or, on the contrary, the quantity of power in excess to be sold. The colors indicate different gravity degrees of this deviation we call *imbalance* (green-yellow-red meaning limited-medium-high imbalance size). Finally, we provided a section called InfoGraphic where we display a comparison between the overall trends for each renewable energy over the hours and also the distributions of the load according to Italian holidays.

2.5.2. Browsable API

Another valuable functionality provided by the Django REST framework is a dedicated REST API. We decided to expose the predictions stored in MySQL DB through a configured API, allowing users to request data based on a specified day and energy.

2.5.3. Newsletter

Lastly, we also implemented a newsletter to forward the data to the users who registered in the designated area of the app. The service can be modified by reconfiguring some parameters (e. g. the news ratio).

3. PIPELINE'S PHASES

In this section, we will describe the pipeline in detail, following the flow of the data from the raw API requests to the result delivery via the web application.

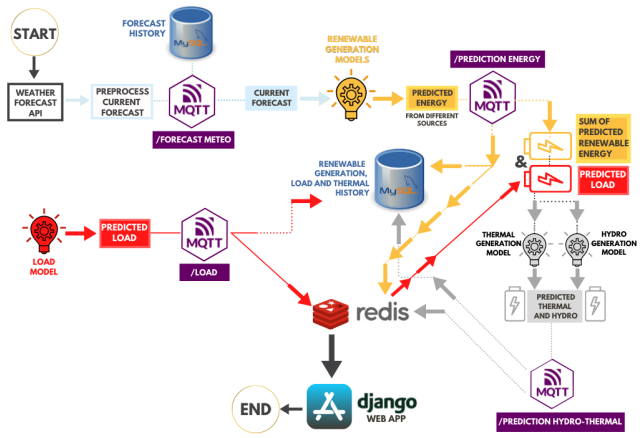

FIGURE 3. Project Pipeline (Actual size image here)

As we mentioned, we decoupled our scripts into small parts in such a way that each service resulted to be the equivalent of an individual process that could be deployed individually. When the whole application is up and running, several processes run at the same time.

We start by describing the processes dedicated to data collection and used to create the history required to train models. Later on, we will discuss data ingestion procedures that begin with the fetching of both meteorological forecasts and data used to predict the energy load.

3.1. Data Collection

3.1.1. Meteorological data

We mainly used two services to collect the essential data for creating our own history to train models. First, we developed a process that does a *get* API request to OpenWheater.com (the website that provides meteorological data) asking for the current weather conditions. We used vanilla python classes to standardize observations. After this pre-processing, data are sent to the MQTT broker at the specific topic Energy/Storico.

The second process is an MQTT-based service, subscribed to Energy/Storico topic. When data are published on this topic by the previous service, the present one saves them into a MySQL DB.

3.1.2. Generation and Load

For energy production and load we relied on Terna, which, as we mentioned in the introduction, provides all the needed data without exposing an API. Therefore, we were forced to manually download those data to store them into our DBs.

3.2. Data Ingestion

The process starts with a *get* API request to the OpenWheater.com website to fetch the weather forecasts of all twenty Italian chief towns from the moment of the call up to 48 hours. We process them via Python: data are first standardized so to reach format homogeneity; then, after having extracted the required information, we average them. The process ends when this information is sent to the MQTT broker, directed to a specific topic called *Energy/ForecastMeteo*

In parallel, a second process starts with the aim of preparing the data required to make predictions of the energy load. As we explained before, the load is highly dependent on sociological aspects such as whether it's holidays or not, as well as on the datetime. Besides, this quantity is unrelated to the meteorological conditions, so it was possible to simply retrieve those data without any external source. When the data are ready, there are sent to the MQTT broker as well and published on a specific topic called *Energy/load*.

3.3. Data Processing

Now, we will describe all the other services that complete the itinerary of the data to the web app. We also built various extra features to make data transfer more manageable for the user (providing options for changing DB and broker, re-training models in light of new data, and many others). Those services, being

extra and not indispensable to complete the flow, are directly described in the guide to run the application.

At this point, we start describing two more services. One is the usual MQTT-based service subscribed to the *Energy/Load* topic that completes the course of load data: when the data arrive from the previous process, this one saves them in a MySQL and Redis DB.

The second process is another MQTT-based service which is instead subscribed to *Energy/ForecastMeteo* topic. This key service takes the incoming meteorological forecasts (just processed) and passes them to the energy renewable models to make predictions for those energy sources that highly depend on the weather. The resultant predictions are sent to the MQTT broker at *Energy/PredictionEnergy* topic.

Another process is the one subscribed to *Energy/PredictionEnergy* topic that operates by receiving the incoming data from the previous step and saving them into the same MySQL and Redis DBs as before. However, this particular service does something more: it takes the previous load data stored within Redis DB and the sum of all the other predicted renewables to feed them to the thermal and hydro models. The last operation is sending the resultant thermal and hydro predictions to the *Energy/PredictionHydroThermal* topic.

The concluding service is subscribed to *Energy/PredictionHydroThermal* topic. It takes those data and stores them in the same MySQL and Redis DBs. Now, the process ends.

3.4. Results and Web App

The endpoint of the process is a web application that graphically displays the results and provides data through a REST API or even through a newsletter, sending predictions at a fixed time by email.

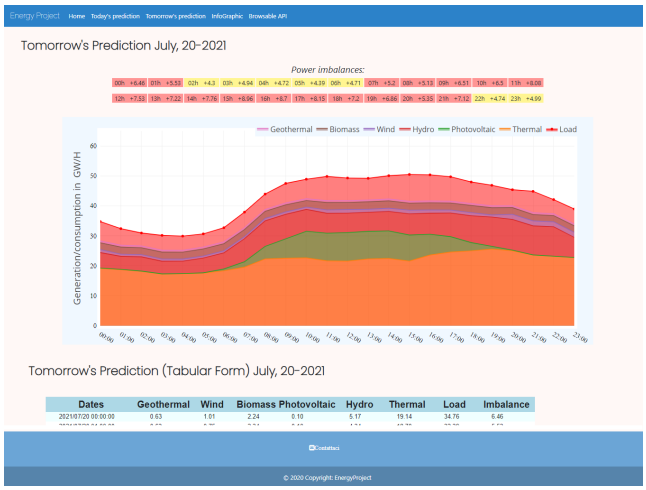

FIGURE 4. Graphical Display of Predictions

You can find the full code, the detailed instructions on how to run the application together with the specific Energy Docker image here: → *GitHubRepository* ←

4. CONCLUSIONS

During this project, we faced different problems. Starting from the data, Terna does not offer any solution to automate the retrieving of renewable energy generation or load data from their website (they also implemented technologies to avoid web scraping). Therefore, as we said, we were forced to manually download those data from the website and upload them into the DB. A possible improvement of this project might be related to the possibility of automating this process. Thus, we could hypothetically build a system that can update itself (learn) every day by comparing its predictions to the real data.

In addition, Terna provides the load measurements for each Italian region, but it does not do so for renewable energy generation. Renewable generation data are national, so we couldn't train different models for different Italian zones, consequently losing much in terms of inference power. We were forced to make an average of the weather data, so to have a single observation that could match with the single generation value.

Another issue concerns the lack of solar radiation data (electromagnetic radiation emitted by the sun). Those data appear to be a very important predictor for photovoltaic and biomass models. We could observe it empirically because OpenWeather.com used to expose a free API providing current solar radiation data together with radiation forecasts until the middle of July, when this service was suddenly not free anymore. Having built the project on those data as well, we searched for a different provider but found none. Hence, we were obliged to change the structure of the project, losing some accuracy in photovoltaic and biomass predictive power.

Ultimately, the usage of MQTT instead of Apache Kafka could represent a structural problem. Apache Kafka seems to provide an overall better solution in terms of quality of service, persistence, and many other factors. However, due to experience and time constraints, we were not in the conditions to implement that specific technology. In any case, MQTT performances were absolutely satisfying for what was required.

REFERENCES

[1] R. donida labati, a. genovese, v. piuri, f. scotti, *Towards the prediction of renewable energy unbalance in smart grids* ieee, 2018 nov 29. pp. 1-5.

[2] https://openweathermap.org/api.

[3] https://www.power-technology.com/features/what-is-geothermal-energy.

[4] https://www.greengeeks.com/blog/hydroelectric-energy.

[5] https://www.engie.com/en/activities/thermal-energy/thermal-power-stations.