TALLER DE LATENCIA II:

Análisis de datos para entender la Nube

Understanding the Cloud through data analysis

Authors: Antonio Bazco-Nogueras, Rita Ingabire, Vincenzo Mancuso
IMDEA Networks Institute, Madrid, Spain
Updated on: November 13, 2023

INSTRUCTOR GUIDE

This document serves as guide for the instructor of the course. We recommend the instructor to carefully read this document before coaching the session with students.

If you have any question or doubt, please contact with the authors by email:

antonio.bazco@imdea.org – rita.ingabire@imdea.org – vincenzo.mancuso@imdea.org

This is guide has been created by the Opportunistic Architectures Lab of IMDEA Networks Institute (Madrid), in the frame of the TelecoRenta Proyect from the UNICO 5G Program.

Plan de Recuperación, Transformación y Resiliencia

Funded by the European Union NextGenerationEU

Welcome to the instructor's guide for the measurements lab *Understanding the Cloud through data analysis*. This document is similar to the corresponding students' guide, with the addition of comments and suggestions for the instructor.

Through this document,[1] the students will learn how we can measure the latency across the whole globe. For that, we explain how to retrieve measurements from the data available through the tool RIPE Atlas, which is a platform from RIPE that allows you to define measurements from any of the thousands of probes available in RIPE Atlas towards any destination (url or IP address).

This document is intended for people that may want to know:

- How does it work the global internet network?
- What is the path that my information follows until it comes back?
- Does it matter where the servers are located in the cloud?
- Can we know *in advance* the performance of the network?

This guide is structured in the following sections:

1) Software Requirements.
2) All you need to know about RIPE Atlas.
3) Retrieving and pre-processing your measurements.
4) Analyzing your measurements.
   In this section, we will see:
   1) Spatio-temporal analysis and
   2) How to use simple Machine Learning methods to predict the latency on Internet.

[1] *Disclaimer:* Some of the content of this document has been retrieved from the web and documentation of RIPE Atlas, https://atlas.ripe.net/. The content is here summarized to facilitate the management of the whole process.

I REQUIREMENTS AND MATERIAL

The first step is to ensure that you have access to all the required materials. We recommend to install the packages in this order: first, the base packages that will allow you to run the software on your computer; second, the Python package manager that enables you to install different python libraries for processing your data; finally, download the experiment files from the public repository. The full list of materials and links is presented next:

Base software:

1) PYTHON 3: The first requirement.
2) PIP: Python Package Manager.
3) JUPYTER LAB & NOTEBOOK: Notebook Interface for Python code.

Python packages:

1) PANDAS: Open-source data analysis and manipulation tool, built on top of the Python programming language.
2) NUMPY: Open-source project that enables numerical computing with Python.
3) MATPLOTLIB: A comprehensive library for creating static, animated, and interactive visualizations in Python.
4) SCIKIT-LEARN: Machine Learning tools in Python.
5) TENSORFLOW: Machine Learning tools in Python.
6) RIPE Atlas Cousteau: A Python wrapper around the RIPE ATLAS API.
7) Other packages: requests, for HTTP messages, datetime for working with time variables, statsmodels for statistical analysis, and graphviz for visualization.

Course scripts:

1) Jupyter Notebook: Contains the core code of the course.
2) RIPE-ATLAS Dataset: Open dataset obtained from RIPE Atlas in JSON format.

Finally, you can also be interested on the site of the main tool that we consider for measuring Internet latency: https://atlas.ripe.net. You will find all the code and supplementary materials in the following GitHub repository: Ripe-ATLAS guide.

II ALL YOU NEED TO KNOW ABOUT RIPE ATLAS

A *How does RIPE Atlas work?*

In brief, RIPE NCC users can request and install (upon acceptance) one of the active RIPE probes in their network. That action allows them to earn credits, which can later be spent to run their own customized experiments. These experiments can use any of the thousands of public probes available through the RIPE Atlas platform.

> *RIPE Atlas is a **collaborative** tool: You must install a probe to earn credits to run your own customized measurements; that is, contributing to the network.*
> *Yet, all measurements are **public and available** even if you do not own any probe.*

You can find a detailed guide on how to use RIPE Atlas in a different course within the same program. If you are interested, please go to this GitHub page.

[*Comment for instructor:*] **The course presented in this manuscript can be considered the continuation of the other course mentioned in the link above. However, they can also be considered completely independent, and the two courses can be followed without the other.**

B *Retrieving data from RIPE Atlas*

You can access all public measurements that have been ever made through RIPE Atlas, and you do not need any login or sign up. You can also follow on-going measurements. They are accessible from this RIPE Atlas site.

[*Comment for instructor:*] **If you have followed the previous course and registered in RIPE Atlas to create your own measurements, you will find a tab called "Mine" that contains the measurements you created.**

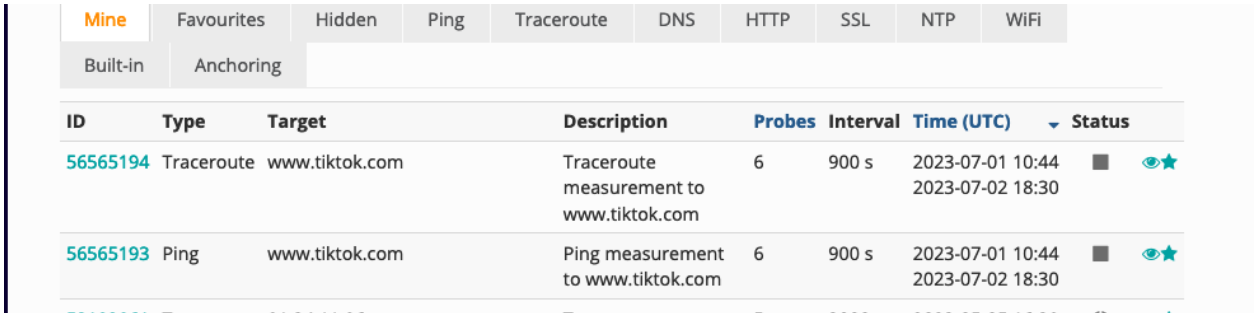

Fig. 1: Interface to browse through all the measurements

The data retrieval is very simple: Click on the measurement corresponding with the experiment you wish to download. This will lead you to an interface where you can specify the data range you are interested in. Click on the download tab and obtain the measurement data file.

III RETRIEVING AND PRE-PROCESSING YOUR MEASUREMENTS

A *Retrieve your data*

The default measurement datasets that will be used in the following experiments are available in JSON format in the public GitHub repository. These data files are enough to proceed with the guide. If you want to use other measurements from RIPE Atlas, you should follow the previous section, and it is sufficient to substitute the name of the files in the python code.

So far, we have explained how to use the RIPE Atlas tool. It is time now to get the most out of it by using the measurements to understand of latency behaves on the worldwide network.

All the data and code required from now on are available at the GitHub repository: the main Python (Jupyter) script named `latency2_cloudDataAnalysis.ipynb`, 6 JSON files with the measurement data, and this guide itself.

Experimental data is stored by RIPE Atlas in a time-series format. If you are interested in knowing more about time series processing techniques, please refer to [1]. The authors of this guide encourage you to choose interpretable machine learning models in your work. Both [1] and [2] are good guides for this topic if you are interested in delving deeper into the topic.

> *Please, from now on, you need to follow the Jupyter notebook named* `latency2_cloudDataAnalysis.ipynb` *and available here. You need to keep following the guide, which gives you details about what we are doing, but you need to run and understand the linked notebook.*

B *Data Cleaning*

The first step in our data analysis is to prepare and clean the data to facilitate our later analysis. In order to view this code open latency2_cloudDataAnalysis.ipynb notebook with the Jupyter Lab Notebook software you have installed.

Once you have evaluated the code to retrieve all the data, we move to clean such data. First, we remove any timeout values or any null values so that those meaningless values do not impact our findings. The data cleaning code is found in Section A.b of the latency2_cloudDataAnalysis.ipynb.

Data Cleaning

```
# Change all null values to NaN
result_df['avg'].replace(-1.0, np.nan, inplace=True)

# Drop all rows with NaN values
result_df = result_df.dropna(subset=['avg'],how='any',axis=0)
```

C *Feature Engineering*

Next, we take our data and transform some of the fields in order to obtain for human-readable values.

Our data's time information is formatted as UNIX timestamps (seconds passed from January 1st, 1970), so we need to convert those values to a more understandable date time format. Probe IDs are in numerical format so you would need to convert this to more a meaningful label. You can also add country names. These are not the only features you can add. Feel free to add as many as you would like. Please, continue in your Jupyter notebook. We reproduce here the code to modify the time values.

Feature Engineering

```
## Changing the time column from epoch to date time format for
    time series processing

new_timestamp = [] # initializing the new array

# populating the new array
for i in result_df['timestamp']:
    my_datetime = datetime.fromtimestamp(i) # transforming the
        timestamp in standard time
    new_timestamp.append(my_datetime) # adding the value to
       the array

# Saving the new time format in the column "new_time"
result_df['new_time'] = new_timestamp
```

You can continue in your notebook and see how the other fields are modified. Finally you visualize the result of our modifications with `result_df.head()`.

We have already prepared our data. Now, we analyze it. Continue in the notebook. This document contains a summary and some key points of the code, so following the notebook allows you to continue in an more deep and interactive way.

IV ANALYZING YOUR MEASUREMENTS

A Spatial and Temporal Analysis

It is always useful to visually inspect the data. You can plot the probability density of all your measurements to see if data is clustered or widespread. This code for this is found in Section B of the notebook skeleton.ipynb.

We can start plotting the histogram of the data to see the probability density function (PDF).

Spatial and Temporal Analysis

```
# Computing the discrete PDF from the histogram

axx = result_df['avg'].hist(density=True, bins=100, alpha =
    0.8)
# bins = Number of points in the histogram.
# alpha = a parameter for color transparency

# We can add information into the graph
axx.set_ylabel("Probability Density Function")
axx.set_xlabel("Latency (ms)")
```

The code above creates the following plot. This graph has a curious shape. Please continue advancing in the notebook to better understand what this graph means. You can also continue through the whole Section A, where other fields are also modified.

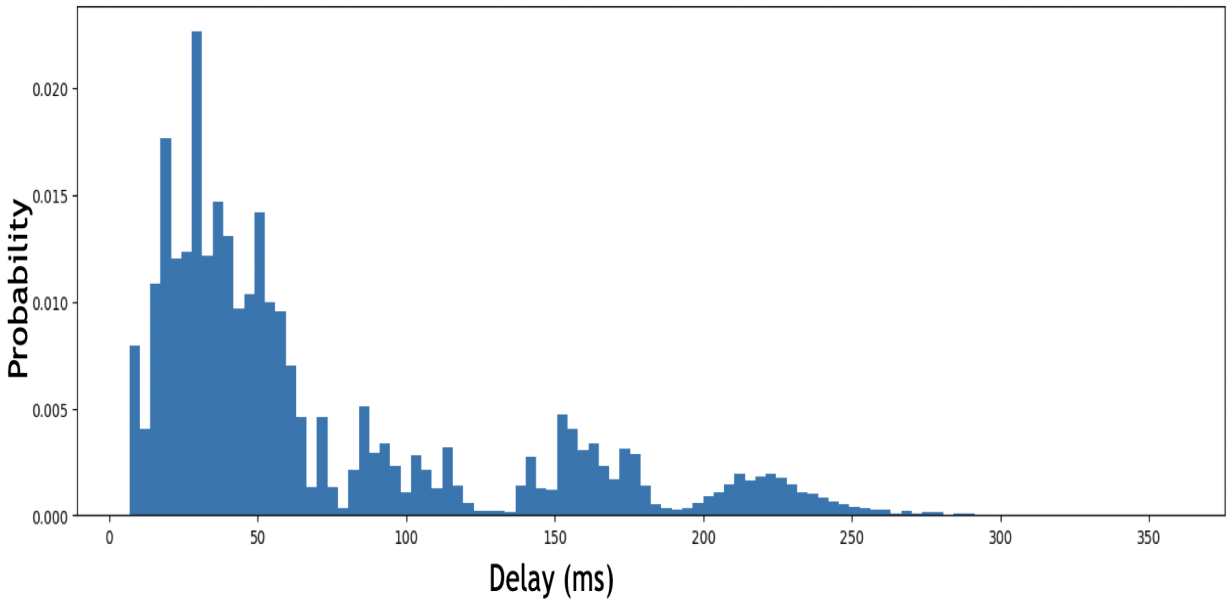

Fig. 2: Probability density plot

B Predicting the data with statistical and machine learning methods

The measurements are time series of latency for different pairs of source and destination and some more metadata. Statistical methods for time series have existed for long time. We start with these methods as first step because they can help us benchmark your predictions. With these benchmarks, we can move on to the more advanced machine learning models.

Below, we give an example of the simplest possible forecast: ***Naive Forecast***. The naive prediction consists on just replicating the previous value as prediction for the next one: We make the assumption that the current latency will be equal to the last value recorded for latency. Please proceed with this part in the notebook.

This is a very basic prediction method but it helps giving us a general idea of what would be a good or bad prediction method. A sample output (not in the notebook) is shown below.

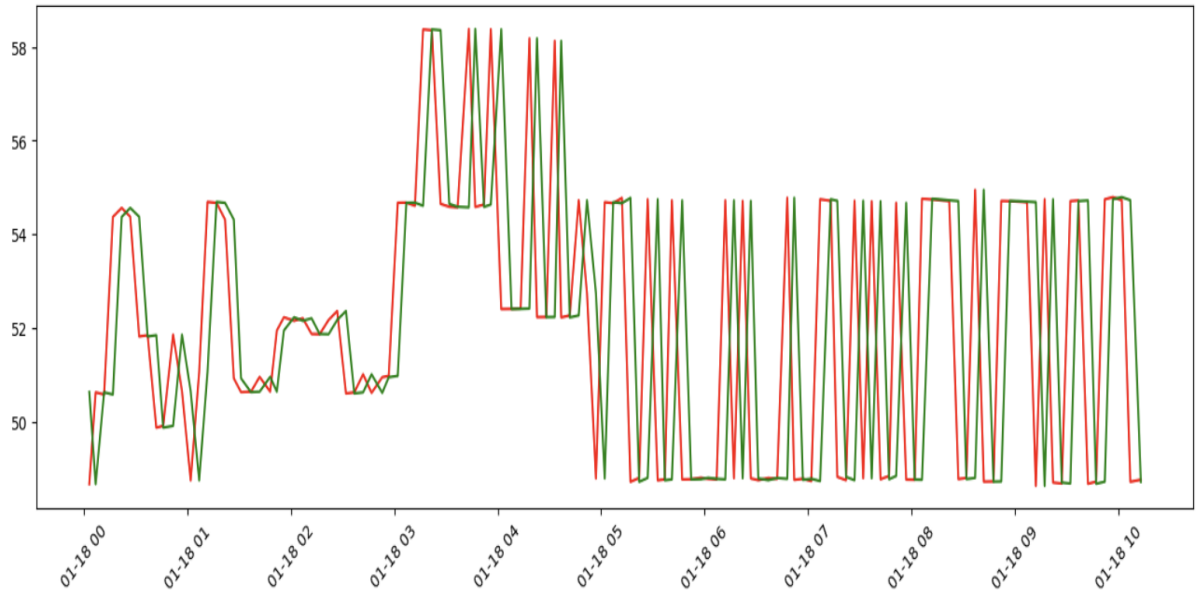

Fig. 3: Naive prediction

We can also use ***Exponential Smoothing*** to make predictions. Results are dependent on a smoothing factor ($0 < \alpha < 1$) applied to previous forecasts which accumulates as we move to the past:

$$y(t+1) = \alpha y(t) + \alpha(1-\alpha)y(t-1) + \alpha(1-\alpha)^2 y(t-2) + \dots$$

Reference [3] provides a very detailed explanation for the existing types of exponential smoothing.

One of the advantages of using Python is the many libraries and packages that are available so we do not need to reinvent the wheel. Applying Exponential smoothing is as simple as follows: (continue in the notebook)

Exponential Smoothing Using Python

```
## Exponential smoothing method

# import the library that allows us to easily apply it
from statsmodels.tsa.api import SimpleExpSmoothing

```

```
# Fitting (adjusting) the data to an exponential smoothing
    approach.
fit1 = SimpleExpSmoothing(result_df['avg']).fit()

# Evaluating the result of the fitting
result_df['Simple-smoothing'] = SimpleExpSmoothing(result_df['
    avg']).fit().fittedvalues
```

Exponential Smoothing is more accurate than Naive forecasting, but it depends on how much we tune α factor. Below, we can find a graph (not in the notebook) showing Exponential Smoothing Forecasting. We use the value of $\alpha = 0.6$ for the factor of smoothing.

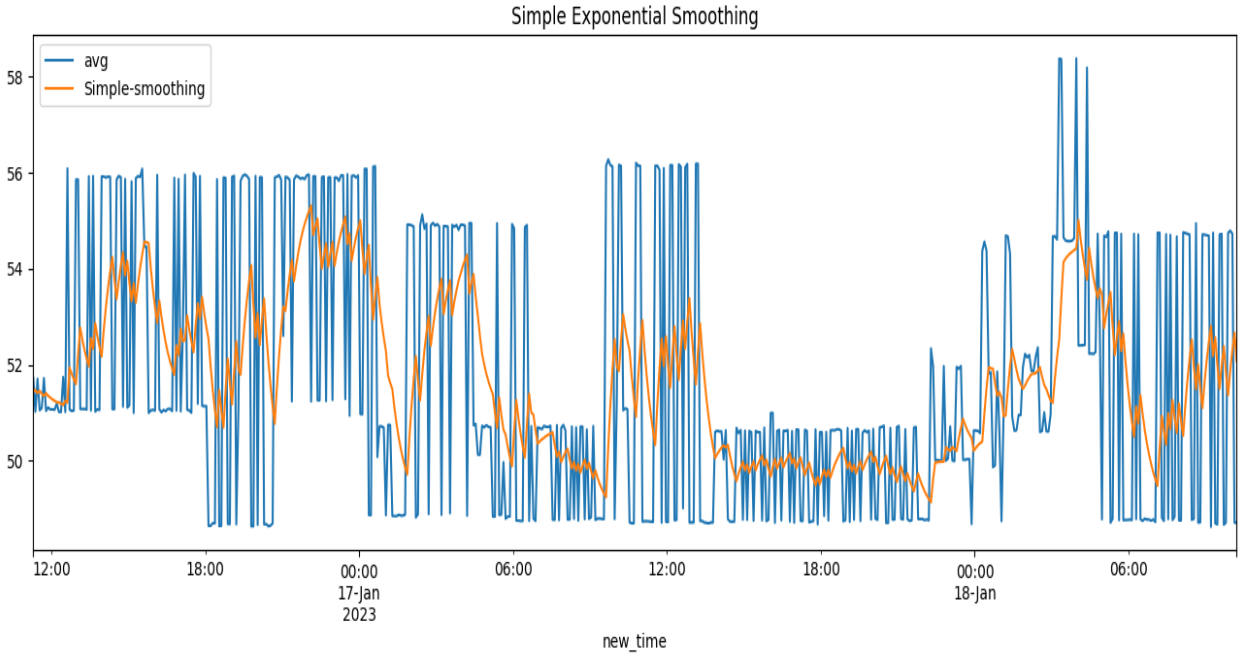

Fig. 4: Exponential Smoothing Modelling

After evaluating very simple statistical methods, we analyze machine learning approaches. Remember to continue executing the notebook while reading this guide.

The first step is to *separate the data in training and test set*. The training set is the data we will use to train the model and let it learn. Then, we evaluate the performance in the test data. This approach is needed in order to avoid that the method ”overfit”, that is, that it adapts too much to the data it sees but it is not able to adapt to new data. Furthermore, it is very important

when considering machine learning algorithms to *normalize the input data*. Proceed to check this section in the notebook.

Once the input data is correctly structured, we can start evaluating different algorithms.

We start with also a simple one: ***Decision Tree Modeling.***

You can find in the notebook the initial steps. But, once the data is correctly set, training the Decision Tree Regressor is as simple as running the following lines:

Decision Tree Code

```
## Import the ML  libraries related to Decision trees
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

## Create a Decision Tree Regressor
model = DecisionTreeRegressor()
## Train the model
model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = model.predict(X_test)
# Evaluate the accuracy of the model
mse = mean_squared_error(y_test, y_pred)
```

In the notebook, we show that you can visualize the tree as decision scheme, as depicted in Figure 5.

Finally, we use neural networks for forecasting. Specifically, we consider a type of Recurrent Neural Network (RNN) called *Long Short-Term Memory* (***LSTM***). LSTM excels in capturing and learning complex sequence patterns so it is ideal for time series forecasting. For this reason, it is commonly found in applications like stock price prediction and weather forecasting.

LSTM is a computationally expensive modelling technique. The training step can be expensive depending on the number of epochs (cycles) you choose to train the model. If the data is simple to predict, you probably do not need this method, but it becomes more and more useful as the data gets involved. You can proceed to run the code in the notebook. There, we show two networks with different complexity.

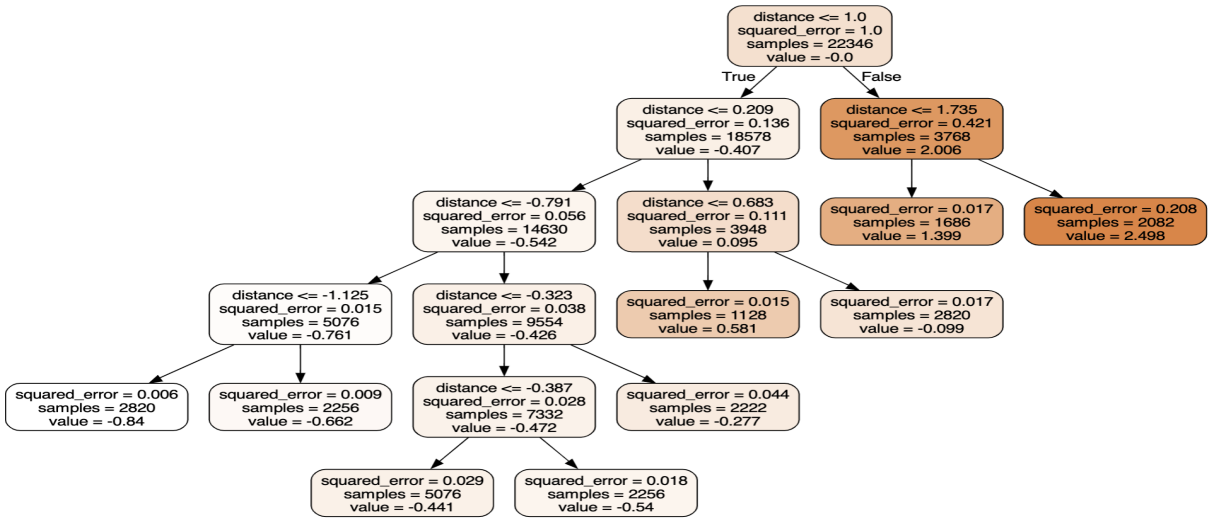

Fig. 5: Decision Tree Graphic

V Closing Remarks & Extension

In this guide, we have explained how we can retrieve, process, analyze, and predict latency data to better understand how Internet operates. If you want to know more about how to create your own experiments, measure latency towards any device on Internet, and more, please check the guide "*RIPE Atlas: Can we find where the Cloud is?*, which can be found in the public GitHub repository.

References

[1] J. Smith. An overview of time series forecasting models. Towards Data Science. [Online]. Available: https://towardsdatascience.com/an-overview-of-time-series-forecasting-models-a2fa7a358fcb

[2] C. Molnar. Interpretable machine learning: A guide for making black box models explainable. [Online]. Available: https://christophm.github.io/interpretable-ml-book/

[3] G. Trubetskoy. (2016) Triple exponential smoothing forecasting. Grisha's Blog. [Online]. Available: https://grisha.org/blog/2016/01/29/triple-exponential-smoothing-forecasting/