

# Practica 6: Predicción de la temperatura en Morelia Michoacán usando Dask

3 de mayo de 2022

Dataset disponible en:  
<https://www.ruoa.unam.mx/index.php?page=estaciones&id=9>

## 1. Objetivo de la práctica.

Crear un modelo de regresión lineal Usando la librería Dask que realice la predicción para la temperatura del clima en °C en la ciudad de Morelia Michoacán.

## 2. Conceptos.

Anteriormente Se han usado librerías básicas de Python como Pandas y scikit learn, ahora se hará una práctica usando una nueva librería de python usada para el procesamiento de grandes volúmenes de datos, Dask.

Dask utiliza estructuras de datos y API de Python existentes para facilitar el cambio entre NumPy, pandas, scikit-learn y sus equivalentes impulsados por Dask.

Generalmente se usan clusters para procesar grandes volúmenes de datos, pero Dask se usa en equipos personales ya que automáticamente examina y gestiona sus recursos.

El dataset consiste en una carpeta que contiene varios archivos csv los cuales representan la información climática de la ciudad de Morelia.

## 3. Herramientas a usar.

Computadora con acceso a internet.  
Los siguientes programas y librerías:

- a. Python versión 3.8.8
- b. Anaconda versión 4.10.1

- c. Jupyter-lab versión 3.0.14
- d. Dask versión 2021.10.0
- e. Dask ML versión 1.9.0
- f. Pandas versión 1.2.4

## 4. Desarrollo.

### 4.1. Entender el problema.

El problema consiste en predecir la temperatura en °C dadas más características climáticas, como la humedad, la hora del día, la velocidad del viento, etc. El procesamiento de los datos debe realizarse usando Dask.

### 4.2. Criterio de evaluación.

En este caso, se propone usar el error medio absoluto (MAE).

### 4.3. Preparar los datos.

- a. Importe las librerías necesarias.

```
import os
import glob
import zipfile
import pandas as pd
import time
import re
import dask.dataframe as dd
from dask.diagnostics import ProgressBar
from dask_ml.linear_model import LinearRegression
from dask_ml.model_selection import train_test_split
from dask_ml.preprocessing import MinMaxScaler
from dask_ml.metrics import mean_absolute_error
```

- b. Descargue los datos del link que se encuentra al inicio de esta práctica. De preferencia use Mozilla firefox o algún servidor FTP como FTPZilla, ya que los datos se descargan a través del protocolo FTP.
- c. Extraer los datos.  
En este caso no es necesario extraer los archivos ya que al descargarlos mediante FTP, estos se descargan sin comprimir.

d. Cargar los datos.

Dask permite cargar muchos archivos csv (que cumplan con el mismo formato) en una sola sentencia, para realizar esto, se le debe pasar como argumento el path en el cual se encuentran todos los archivos y añadir al final "\*.csv", esto se puede interpretar como cargar todos los archivos csv.

```
Path = "/home/michell21/Datasets/Ruoa_Dask_minuto/"
df = dd.read_csv(Path+'*.csv')
```

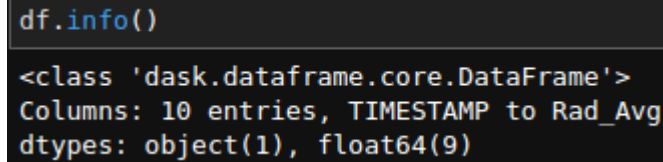
Código 1

e. Explorar los datos.

Al usar dask en lugar de pandas para cargar los datos, estos también son un DataFrame, pero no se pueden usar exactamente los mismos métodos que se usan en pandas.

Use `info()` para ver información sobre el DataFrame. [Ver la Figura 1]

```
house.info()
```



```
df.info()
<class 'dask.dataframe.core.DataFrame'>
Columns: 10 entries, TIMESTAMP to Rad_Avg
dtypes: object(1), float64(9)
```

Figura 1: Función info()

En este caso no se pudo saber si hay registros vacíos o no ya que dask no muestra la información igual que pandas, en su lugar muestra el número de columnas que componen el DataFrame.

Si se quiere saber el contenido de las columnas es un DataFrame de Dask, se debe agregar a la instrucción `.compute()` para indicar que se quiere hacer alguna operación sobre los datos. Por ejemplo, para ver la información en la columna *Temp\_Avg* se debe aplicar el siguiente comando.

```
df["Temp_Avg"].describe().compute()
```

Código 2

Y el resultado se ve así:

f. Preprocesamiento de los datos.

```
df["Temp_Avg"].describe().compute()

count      3.225764e+06
mean       1.745081e+01
std        4.972629e+00
min        5.080000e-01
25%        1.571000e+01
50%        1.924000e+01
75%        2.479000e+01
max        3.369000e+01
Name: Temp_Avg, dtype: float64
```

Figura 2: Función compute() aplicado a describe()

En caso de que exista valores nulos en la etiqueta a predecir (*Temp\_Avg*), estos registros se eliminan para evitar algún sesgo en el entrenamiento y la futura predicción. También se rellenan los demás datos nulos con 0 Para eliminar los registros donde los valores en la etiqueta sean nulos y rellenan los demás con 0, se usa la siguiente instrucción.

```
df = df.dropna(subset=["Temp_Avg"])
df = df.fillna(0)
```

### Código 3

Hay que recordar que se busca predecir la temperatura dados diferentes factores, entre ellos el mes del año y si es de día o de noche, pero esa información no se incluye como tal en el dataframe. Sin embargo, en la columna *TIMESTAMP* viene información sobre la fecha y hora en que se tomó la muestra climatológica, el formato de la misma es *yyyy-mm-dd HH:MM:SS*.

Con esa información se puede extraer tanto el mes del año como si es de día o de noche y agregarlo en otra columna.

Para Extraer la información se usa una función la cual será aplicable a cada una de las particiones del DataFrame de Dask con la función `lambda`, la cual funciona como un optimizador a la hora de aplicar funciones y operaciones en grandes volúmenes de datos.

El código a aplicar es el siguiente

```
def month(TimeStamp):
    date = re.findall(r'\d{2}', TimeStamp)[0]
    date = date.replace("-", " ")
```

```

    return int(date)

def day_night(TimeStamp):
    hour = re.findall(r'\d{2}', TimeStamp)[0]
    hour = hour.replace("_", "")
    hour = int(hour)
    if hour < 12:
        return 0 #AM
    return 1 #PM

def find_month(df):
    df["Mes"] = df.apply(lambda Row: month(Row["TIMESTAMP"]), axis=1)
    df["Dia/Noche"] = df.apply(lambda Row: day_night(Row["TIMESTAMP"]), axis=1)
    #df1 = dd.from_pandas(df, npartitions=76)
    return df

```

#### Código 4

Este código se tiene que aplicar directamente sobre los datos, pero esto los transformará en un objeto de pandas, así que una vez que las funciones sean aplicadas se debe transformar de nuevo en un dataframe de dask.

```

df["Mes"] = 0
df["Dia/Noche"] = 0
df = df.map_partitions(find_month, meta=df).compute(scheduler='processes')
df = dd.from_pandas(df, npartitions=76)

```

#### Código 5

El resultado de este código se puede ver en la figura 3.

|   | TIMESTAMP           | Temp_Avg | RH_Avg | WSpeed_Avg | WSpeed_Max | WDir_Avg | WDir_SD | Rain_Tot | Press_Avg | Rad_Avg | Mes | Dia/Noche |
|---|---------------------|----------|--------|------------|------------|----------|---------|----------|-----------|---------|-----|-----------|
| 0 | 2020-06-01 00:00:00 | 16.15    | 70.69  | 1.004      | 1.116      | 100.300  | 4.682   | 0.0      | 806.000   | -1.337  | 6   | 0         |
| 0 | 2020-11-01 00:00:00 | 16.89    | 87.80  | 1.702      | 1.946      | 242.500  | 16.940  | 0.0      | 809.553   | -1.429  | 11  | 0         |
| 0 | 2016-03-01 00:00:00 | 14.39    | 75.55  | 1.002      | 0.000      | 262.000  | 8.470   | 0.0      | 806.000   | 0.000   | 3   | 0         |
| 0 | 2021-02-01 00:00:00 | 14.17    | 56.79  | 2.154      | 2.529      | 267.800  | 7.473   | 0.0      | 808.148   | -1.106  | 2   | 0         |
| 0 | 2020-12-01 00:00:00 | 16.63    | 70.85  | 2.391      | 3.379      | 2.498    | 5.615   | 0.0      | 807.000   | -1.475  | 12  | 0         |

Figura 3: Datos limpios

- g. Dividir los datos en Train y test (X y y).

Hay que recordar que el objetivo es predecir la temperatura, por lo tanto, la variable  $y$  en este caso es la columna *Temp\_Avg*.

De las columnas restantes, se puede descartar *TIMESTAMP* debido a que son valores únicos que representan la fecha y la hora en que se tomó la muestra, por lo tanto no aporta información relevante.

Usando el siguiente código, se dividen los datos en  $X$  y  $y$ .

```
y = df["Temp_Avg"]
X = df.drop(["TIMESTAMP", "Temp_Avg"], axis=1)
```

### Código 6

Los datos divididos se muestran en la figura 4

|   | RH_Avg | WSpeed_Avg | WSpeed_Max | WDir_Avg | WDir_SD | Rain_Tot | Press_Avg | Rad_Avg | Mes | Dia/Noche |
|---|--------|------------|------------|----------|---------|----------|-----------|---------|-----|-----------|
| 0 | 70.69  | 1.004      | 1.116      | 100.300  | 4.682   | 0.0      | 806.000   | -1.337  | 6   | 0         |
| 0 | 87.80  | 1.702      | 1.946      | 242.500  | 16.940  | 0.0      | 809.553   | -1.429  | 11  | 0         |
| 0 | 75.55  | 1.002      | 0.000      | 262.000  | 8.470   | 0.0      | 806.000   | 0.000   | 3   | 0         |
| 0 | 56.79  | 2.154      | 2.529      | 267.800  | 7.473   | 0.0      | 808.148   | -1.106  | 2   | 0         |
| 0 | 70.85  | 2.391      | 3.379      | 2.498    | 5.615   | 0.0      | 807.000   | -1.475  | 12  | 0         |

```
Dask Series Structure:
npartitions=76
0          float64
574          ...
...
43677          ...
44639          ...
Name: Temp_Avg, dtype: float64
Dask Name: getitem, 152 tasks
```

Figura 4: Datos divididos en  $X$  y  $Y$

#### h. Escalar los datos en $X$ .

Escalar datos en dask es igual que en pandas, pero en lugar de usar la biblioteca scikit-learn, dask\_ml contiene un modulo de preprocesamiento en el cual viene [MinMaxScaler](#).

Al igual que en pandas, MinMaxScaler toma como argumento el dataframe y regresa un arreglo del mismo con los datos escalados.

Lo anterior se logra usando el siguiente código:

```
scaler = MinMaxScaler()
X_scaler = scaler.fit_transform(X)
```

### Código 7

Para poder dividir un dataframe de dask, usando `train_test_split` de `dask_ml.model_selection` primero se deben extraer los valores de este y convertirlos en un array. De esta forma, la división de datos podrá aplicarse correctamente a cada partición del dataframe.

Al usar el siguiente código se puede aplicar lo mencionado anteriormente.

```
X_scaler = X_scaler.to_dask_array()
y = y.to_dask_array()
X_train, X_test, y_train, y_test =
train_test_split(X_scaler.compute_chunk_sizes(),
y.compute_chunk_sizes(), test_size = 0.3, shuffle=True)
```

### Código 8

Al aplicar esta función, los datos ya no son visibles, como se ve en la figura 5, pero este es el formato que se necesita para realizar entrenamiento en dask.

|              | Array         | Chunk         |            |
|--------------|---------------|---------------|------------|
| <b>Bytes</b> | 172.27 MiB    | 2.27 MiB      |            |
| <b>Shape</b> | (2258000, 10) | (29754, 10)   | 10 2258000 |
| <b>Count</b> | 608 Tasks     | 76 Chunks     |            |
| <b>Type</b>  | float64       | numpy.ndarray |            |

Figura 5: X train en dask

#### 4.4. Construir el modelo.

La librería de `dask_ml` viene precargada con modelos de ML, al igual que `scikit-learn`, por lo tanto solo basta con cargar el modelo de regresión lineal y ponerlo a entrenar.

Al usar `ProgressBar` podemos ver el progreso del entrenamiento. Hay que recordar que el entrenamiento realiza un ciclo por cada partición de dataframe de `dask`, es decir que si se tienen 76 particiones, el entrenamiento realizará 76 ciclos.

Se usa el siguiente código para llamar al modelo de regresión y entrenarlo.

```
lr = LinearRegression()
with ProgressBar():
    lr.fit(X_train, y_train)
```

Código 9

El progreso del entrenamiento se vería como en la figura 6

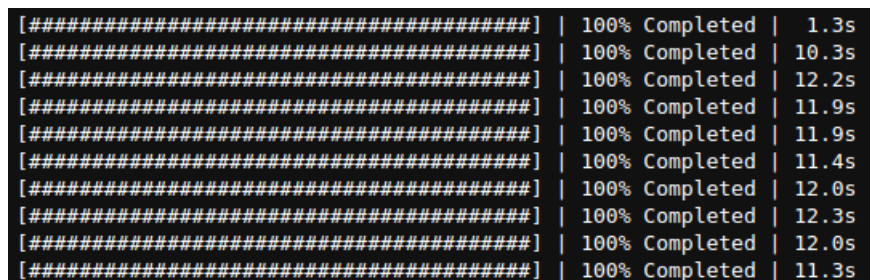


Figura 6: Visualización del entrenamiento usando `ProgressBar`

#### 4.5. Análisis de errores.

Para obtener predicciones de un modelo en `dask`, se usa `predict` como se hace en `scikit-learn`. Hay que recordar que el entrenamiento regresa un arreglo de `dask`. Así que si se quiere comparar resultados, se debe convertir este arreglo en un dataframe de `dask`.

Los resultados se obtienen con este código.

```
y_pred = lr.predict(X_test)
#Los arreglos se transforman en un dataframe
y_pred_df = y_pred.to_dask_dataframe()
y_test_df = y_test.to_dask_dataframe()
```

Código 10



|                |       |                |           |
|----------------|-------|----------------|-----------|
| 0              | 14.19 | 0              | 13.470033 |
| 1              | 20.04 | 1              | 14.790290 |
| 2              | 16.28 | 2              | 16.708427 |
| 3              | 3.47  | 3              | 12.621100 |
| 4              | 9.75  | 4              | 13.936094 |
| dtype: float64 |       | dtype: float64 |           |

Figura 7: Y verdadera (izquierda) Y predicha (derecha)

Los dataframes se verían como se muestra en la figura 7

De esta manera, se puede usar el MAE de la misma forma que se ha hecho en las practicas anteriores.

```
mean_absolute_error(y_test , y_pred)
```

#### Código 11

El resultado del [Código 11](#) es 2.35, esto representa que el modelo tiene un margen de error de 2 °C lo cual es un resultado aceptable ya que en el mundo real las predicciones no suelen ser acertadas y fallan por algunos grados.

## 4.6. Implementación.

Ver el Notebook de Jupyter llamado *P6-Ruoa-Regresion.ipynb*