Anexo I. Flujo de datos

Vamos a explicar con más detenimiento el flujo de datos dentro de la aplicación, desde que los descargamos de la web, hasta que los introducimos al modelo. Para esto, tomaremos uno de los dos ejemplos expuestos en el trabajo, y seguiremos todos los pasos hasta la llamada al algoritmo del modelo.

Usaremos como ejemplo los datos del número de ocupantes de los taxis de Nueva York. De la página web, nos descargamos un archivo CSV, o "*Comma-Separated Values*", un tipo de fichero usado mucho para almacenar datos. La codificación en este tipo de formato es muy sencilla: se elige un carácter delimitador (usualmente, ',' o ';'), y se separan los distintos datos de una medida usando ese separador. Las distintas medidas se almacenan en distintas líneas del fichero (Figura 1). Se usa la primera línea para indicar qué dato corresponde a cada columna.

```
timestamp,value
2014-07-01 00:00:00,10844
2014-07-01 00:30:00,8127
2014-07-01 01:00:00,6210
2014-07-01 01:30:00,4656
```

Figura 1: Ejemplo datos formato CSV. Fuente: Propia

Una vez que tenemos los datos guardados en un fichero, necesitamos cargarlos en memoria para poder trabajar con ellos. Por suerte, el lenguaje Python con el que estamos trabajando tiene librerías para la lectura de este tipo de ficheros, sólo teniendo que configurar el tipo de datos que hay dentro del CSV (Figura 2).

```
import csv # Librería para trabajar con ficheros .CSV
import datetime # Librería para trabajar con fechas

fileName = './Data/nyc_taxi.csv' # Declaramos el nombre del fichero a leer
readedData = []

with open(fileName, 'r') as file: # Abrimos el fichero como stream para acceder a los datos
    reader = csv.DictReader(file) # Creamos un lector de CSV
    firstLine = True # Almacenamos un valor para no leer la primera línea
    for row in csv_reader:
        if firstLine:
            firstLine = False # Ignoramos la primera línea
        else:
            date = datetime.datetime.strptime(row["timestamp"], "%Y-%m-%d %H:%M:%S")
            value = int(row["value"])

            data = { # Almacenamos los datos en una variable
                "date": date,
                "value": value
            }
            readedData.append(data)
```

Figura 25: Código lectura de fichero CSV. Fuente: Propia

Una vez que ya tenemos los datos almacenados, es necesario convertirlos a un sistema de datos que nos permita enviarlos a la red (Figura 3).

```
def encodeScalar(self, input, output):
    # Primero, obtenemos el primer bit activo. Esta es una función externa.
    firstBit = self.getFirstBit(input) # Self referencia a funcinoes y variables de esta misma clase

    # Comprobación de error
    if firstBit is None:
        output[0:self.bucketLen] = 0 # Hacemos un array de la longitud del cubo vacio

    else:
        output[0:self.bucketLen] = 0
        # Obtenemos el último bit activo
        lastBit = firstBit + self.activeBits

        # Hay que tener cuidado, ya que el mínimo o el máximo se pueden pasar de tamaño
        if lastBit >= self.bucketLen:
            output[:(lastBit - self.bucketLen + 1)] = 1
            lastBit = self.bucketLen - 1
        if firstBit < 0:
            output[self.bucketLen + firstBit:self.bucketLen] = 1
            firstBit = 0

        output[firstBit:lastBit + 1] = 1

def getFirstBit(self, data):
    # Función que calcula el primer bit activo
    centerBit = int((data - self.minValue) * self.internalNumber / self.activeBits) / self.padding
    firstBit = int(centerBit - 0.5 * self.activeBits)
    return firstBit
```

Figura 3: Código codificación números. Fuente: Propia

Con estos datos, ya podemos llamar a la red y empezar con el entrenamiento. La llamada a la red es muy sencilla, ya que los datos los tenemos en el formato que entiende la red, y los valores necesarios para su creación los tenemos ya calculados (Figura 4).

```
from nupic.frameworks.opf.model_factory import ModelFactory
from nupic.frameworks.opf.common_models.cluster_params import getScalarMetricWithTimeOfDayAnomalyParams


# Calculamos el rango de valores para usar en la codificación, son necesarios para el modelo
rangePadding = abs(self.inputMax - self.inputMin) * 0.2

# El propio modelo tiene una función que nos coloca todas las funcioens necesarias
modelParams = getScalarMetricWithTimeOfDayAnomalyParams(
  metricData=[0],
  minVal=self.inputMin-rangePadding,
  maxVal=self.inputMax+rangePadding,
  minResolution=0.001,
  tmImplementation = "cpp" # Indicamos que queremos usar la implementación en C++ (más rápida)
)["modelConfig"]

# Y creamos el modelo
self.model = ModelFactory.create(modelParams)
```

Figura 4: Creación del modelo. Fuente: Numenta