Práctica Final

Minería de datos

2016-2017

Nicolás González Munar

Introducción

En esta práctica el objetivo era conseguir una predicción del consumo eléctrico que se realizará dada una determinada climatología. Para ello, debe extraerse dicha información de diferentes fuentes (la web de Red Eléctrica Española para el consumo energético, y la Agencia Española de Meteorología para la climatología). Una vez obtenida esa información, debe generarse la predicción, por ejemplo con el uso de una red neuronal.

Por tanto, en este proyecto se distinguen 3 apartados importantes, que consolidan proyectos por separado. Dichos proyectos, y sus ubicaciones son las siguientes

1. Scrap a la web de Red Eléctrica Española, en la carpeta ***Scraper***.
2. Peticiones sucesivas a la API del AEMET, en la carpeta ***OpenData API***.
3. Red neuronal, en la carpeta ***Neural Network***.

Las 2 primeras etapas, de captura de datos, guardan los datos en formato JSON cada uno en un archivo, donde cada JSON representa los datos para 1 día en concreto, y ambos ficheros se copian en la carpeta ***Extracted data***. Luego, esos datos se hacen pasar por la red neuronal, se entrena, y se prueba.

Todas las partes se han realizado en **Python**. Esta decisión ha sido tomada debido a que Python es un buen lenguaje para scripting, que es el tipo de aplicaciones que más se asemeja a lo que buscamos con las 2 primeras partes de la práctica.

Como los datos se extrajeron en formato JSON, el lenguaje a usar para la red neuronal era indiferente. Así pues se hizo una búsqueda de una librería que resultase sencilla de usar y diera buenos resultados, y se realizó una búsqueda para C++, C#, Visual Basic y Python. La librería más atractiva de las encontradas fue **PyBrain**, para Python (http://pybrain.org/). Ofrece una red neuronal muy sencilla de utilizar (como podrá verse más adelante), así como multitud de opciones para hacer estructuras más complejas si se requiere.

Por tanto, en este documento podrán encontrarse 5 apartados más; uno para cada uno de los subproyectos mencionados, uno con pruebas y sus análisis, y un último con las conclusiones.

Scraper

Este proyecto tiene como función obtener los datos de consumo energético diario en una comunidad autónoma. Esta información nos la brinda “Red Eléctrica Española”. Concretamente, podemos acceder a la información de un día concreto a través de la url:

http://www.ree.es/es/balance-diario/baleares/YYY/MM/DD

Dado que lo que queremos es entrenar una red neuronal, cuanto mayor sea el conjunto de datos, más capacidad de entrenamiento tendrá dicha red y por tanto más precisa será con las predicciones. Por este motivo, se ha buscado cuál es la fecha más antigua para la cual tienen datos y es **2012/01/01**. Este es, por tanto, el punto de partida de la captura de datos.

A nivel técnico, se ha utilizado la archiconocida librería ***Scrapy***, librería ampliamente documentada y de la cual se puede encontrar información muy fácilmente gracias al uso de *XPath* para recorrer el DOM de las webs. Esta librería permite definir diferentes *spiders*, que con simples funciones pueden ponerse en marcha de forma sencilla. En concreto hemos definido el siguiente spider:

```
class DataSpider(scrapy.Spider):
    name = "data"

    def start_requests(self):
        for yy in range(2012, 2017):
            for m in range(1, 13):
                mm = str(m)
                mm = mm if (len(mm) > 1) else '0' + mm
                for d in range (1, 29):
                    dd = str(d)
                    dd = dd if (len(dd) > 1) else '0' + dd
                    yield scrapy.Request(
                        url = "http://www.ree.es/es/balance-diario/baleares/" + str(yy) + "/" + mm + "/" + dd,
                        callback = self.parse
                    )

    def parse(self, response):
        l = ItemLoader(item = Data(), response = response)
        #Date
        l.add_xpath('date', '//h1/text()')
        #Generacion
        l.add_xpath('generacion', '//th[contains(text(), "Generac") and @class = "texto3"]/following-sibling::td[1]/text()')
        return l.load_item()
```

Como puede verse, se itera desde el año 2012 hasta el 2016, los meses del 1 hasta el 12, y, por sencillez, los días del 1 al 28. Se ha hecho así por no ensuciar un algoritmo sencillo por un par de datos que representan una porción muy pequeña de todos los que obtendremos sin ser realmente importantes.

La información que se recoge es la fecha y la “Generación” de energía. Para dejar esta información más accesible a la red neuronal, antes de convertirla a JSON y escribirla, se pone la fecha en un formato más limpio:

```
#Remove day of week from date
item['date'] = item['date'].split(', ', 1)[-1]
line = json.dumps(dict(item), ensure_ascii = False) + "\n"
self.data.write(line)
```

OpenData API

La finalidad de este script es recoger la información climatológica acontecida en los días a estudiar. Esto lo hacemos gracias a la API "*Open Data*" de la Agencia Española de Meteorología. Esta API permite solicitar la información climatológica que capturó una determinada estación un determinado día con una simple llamada GET, indicando en la cabecera de la misma nuestra *api-key*.

Como lo que queremos es la información para Baleares, se ha escogido la estación "Palma de Mallorca - Aeropuerto" (con id B278). Solo queda realizar una llamada para cada día, lo cual puede verse en el siguiente código:

```
data = codecs.open('climData.json', 'w', encoding = 'utf-8')
#Loop through days
for yy in range(2012, 2017):
    for m in range(1, 13):
        mm = str(m)
        mm = mm if (len(mm) > 1) else '0' + mm
        for d in range (1, 29):
            dd = str(d)
            dd = dd if (len(dd) > 1) else '0' + dd
            #URL construction
            date = str(yy) + "-" + mm + "-" + dd
            fechaini = "/fechaini/" + date + "T00:00:00UTC"
            fechafin = "/fechafin/" + date + "T23:59:59UTC"
            url = host + fechaini + fechafin + estacion
            #AEMET API call (GET)
            response = requests.get(url, headers = headers, verify = False)

            #Unserialize data into JSON
            responseData = json.loads(response.text)

            #Actual Data information
            response = requests.get(responseData['datos'], headers = headers, verify = False)

            #Convert data into JSON and write it
            line = json.loads(response.text)[0]
            line = json.dumps(dict(line), ensure_ascii = False) + "\n"
            data.write(line)
            print(line)

            #For respecting qpm calls, wait
            time.sleep(3)
        time.sleep(5)
    time.sleep(5)
data.close()
```

Hay varios detalles a destacar. El primero de ellos es que la respuesta de la primera llamada devuelve información referente a la misma (estado de la transacción, código de error HTTP), y un enlace en el cual descargar un JSON con la información climatológica. Esto hace que cada día requiera 2 llamadas para obtener la información.

Otro detalle a destacar es que la OpenData API tiene un máximo de llamadas por minuto, y para respetar dichos tiempos, se han añadido ciertos sleeps. Hay que decir que el proceso de obtención de los datos, pues, tarda bastante tiempo en finalizar (~2h).

Neural Network

En este script se procesan los JSON generados anteriormente y se utilizan para entrenar una red neuronal, la cual puede usarse a posteriori para realizar predicciones. Merece la pena entretenerse un poco más en explicar el código contenido en el mismo.

```
#Create neural netwotk: 4 inputs, 30 hidden layers, 1 output
net = buildNetwork(4, 30, 1, bias = True)

#Create data structure: [tmax, tmin, presMax, presMin] -> [consumo]
ds = SupervisedDataSet(4, 1)
```

Primero, instanciamos la red neuronal con la función *buildNetwork*. La instanciaremos para 4 inputs, 1 output, y 30 neuronas en su *hidden layer*. Después instanciaremos el *dataSet* que utilizaremos para entrenarla. Como puede aprecierse, se ha elegido coger como inputs: Temperatura máxima y mínima y presión máxima y mínima, para intentar obtener una representación más fidedigna que con solo temperaturas o valores medios. Por otro lado, decir que se le indica a la red neuronal que utilice *bias*, pues los resultados son más ajustados.

Después procedemos a procesar los datos recolectados previamente:

```
#Prepare data.json (Create a dictionary indexed by date):
consData = {}
datajson = open("../Extracted data/data.json")
months = {"enero":"01", "febrero":"02", "marzo":"03", "abril":"04", "mayo":"05", "junio":"06",
          "julio":"07", "agosto":"08", "septiembre":"09", "octubre":"10", "noviembre":"11", "diciembre":"12", }
for line in datajson:
    data = json.loads(line)
    date = data['date'].split(' ')
    if len(date[0]) == 1: date[0] = "0" + date[0]
    consData[date[2] + "-" + months[date[1]] + "-" + date[0]] = float(data['generacion']) / 1000
datajson.close()

#Read climData and insert examples to the network:
climData = open("../Extracted data/climData.json")
for line in climData:
    clim = json.loads(line)
    tmax = float(clim['tmax'].replace(',', '.'))
    tmin = float(clim['tmin'].replace(',', '.'))
    #Normalize pressure values
    presMax = float(clim['presMax'].replace(',', '.')) /100
    presMin = float(clim['presMin'].replace(',', '.')) /100
    print("(" + str(tmax) + ", " + str(tmin) + ", " + str(presMax) + ", " + str(presMin) + ") -> " + str(consData[clim['fecha']]))
    ds.addSample([tmax, tmin, presMax, presMin], consData[clim['fecha']])
climData.close()
```

Lo primero que hay que darse cuenta es que los formatos de fecha son diferentes, por lo que será necesario tratar las fechas de alguno de los archivos. Además, si nos fijamos, los datos recogidos por scrapping no han quedado ordenados por fecha. Esto se debe a que, por el hecho de funcionar a través de *callbacks*, pueden haber quedado las peticiones desordenadas.
Es por este motivo que se hace previamente un recorrido sobre este fichero y se crea un “diccionario”, es decir, línea a línea se da formato YYYY-MM-DD a la fecha, que es el

formato que tienen las fechas en el otro fichero y se añade una entrada Key-Value que sea Fecha-Generación. También se aprovecha este momento para normalizar los valores de generación para que se asemejen más a los valores climatológicos (dividiéndolos entre 1000).

Justo después se lee el otro fichero. Se cogen los JSONs con la información climática, se obtienen las temperaturas y presiones (normalizando estas últimas dividiéndolas entre 100), y se obtiene la generación de ese día a través de su fecha, y se almacena dicha información en el *dataSet* previamente definido.

```
#Train the network
trainer = BackpropTrainer(net, ds, momentum = 0.1, verbose = True, weightdecay = 0.01)
if(i in "Ss"):
    valid = False
    while(not valid):
        i = raw_input('How many epochs do you want the network to train?\n')
        if i.isdigit(): valid = True
    i = int(i)
    error = trainer.trainEpochs(i)
else:
    print("This operation can take a while. Please, be patient.")
    error = trainer.trainUntilConvergence()

print("Network TRAINED!\n")

#Test the network
valid = False
while(not valid):
    print("\n\nSelect a testing option:\n     1. Test a value.\n     2. Test file values (ones used to train the NN).\n     3. Exit.")
    i = input()
    if(i == 1):
        i = input('\nIntroduce an output to test or [] to exit (input format: "[tmax, tmin, presMax/100, presMin/100]"): ')
        res = net.activate(i) * 1000
        print(res)

    elif(i == 2):
        i = raw_input('Please, name the file where you want the results to be written in: ')
        climData = open("../Extracted data/climData.json")
        results = codecs.open(i, 'a', encoding = 'utf-8')
        for line in climData:
            clim = json.loads(line)
            tmax = float(clim['tmax'].replace(',', '.'))
            tmin = float(clim['tmin'].replace(',', '.'))
            presMax = float(clim['presMax'].replace(',', '.')) /100
            presMin = float(clim['presMin'].replace(',', '.')) /100
            result = str(net.activate([tmax, tmin, presMax, presMin]) * 1000) + "\n"
            results.write(result)
        results.close()
        climData.close()
        print("\nDONE. You can find the results on " + i + " file.")

    elif(i == 3):
        valid = True
        break
```

El código que sigue queda bastante oculto entre interacciones con el usuario. Primero creamos el *trainer*, basado en *Backpropagation*. Se han probado valores de *momentum* y *weightdecay* y los que mejor resultado han dado han sido los que pueden verse. Por otro lado, el *verbose* nos permite ver los errores que quedan en cada una de las fases del entrenamiento.

Básicamente se han utilizado 2 métodos de entrenamiento: *trainEpochs(n)* y *trainUntilConvergence()*. El primero permite realizar n fases de entrenamiento, mientras que el segundo entrena hasta reducir el error a 0. Gracias al flag *verbose* activado anteriormente, ambos métodos imprimirán los errores que obtengan tras cada fase. Cabe destacar que el segundo método necesita un elevadísimo tiempo, si no infinito, para reducir a 0 el error.

Una vez entrenada la red neuronal, pueden probarse valores y ver qué resultado daría con la función *activate(x)*. Para ello han sido creados 2 modos de testeo: probar para un conjunto de valores, o probar para todos los valores usados para entrenar. Esta opción vuelca los resultados en un fichero, lo cual será útil para su análisis posterior.

Un ejemplo de ejecución:

```
Do you want a simple or a complete training? (For one epoch or until convergence) (S/C) s
How many epochs do you want the network to train?
5
Total error:  3.46724210239
Total error:  3.39633255439
Total error:  3.38722411445
Total error:  2.88865340438
Total error:  2.88318979464
Network TRAINED!



Select a testing option:
     1. Test a value.
     2. Test file values (ones used to train the NN).
     3. Exit.
1

Introduce an output to test or [] to exit (input format: "[tmax, tmin, presMax/100, presMin/100]"): [19, 4, 10.26, 10.23]
[ 12538.67105556]
```

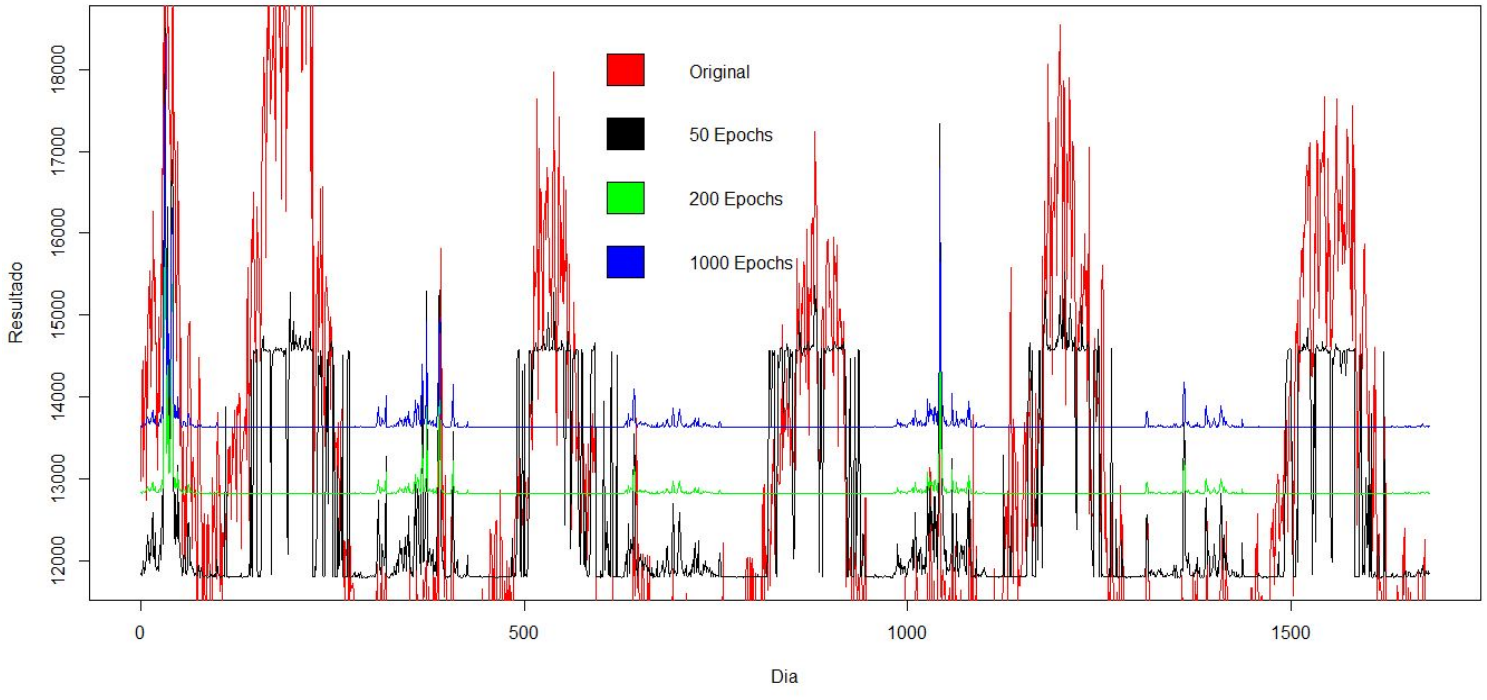
Los valores de error son elevados, pero hay que tener en cuenta que los valores no están normalizados entre 0 y 1. Además, se introduce un input similar al de día 1-1-2012: "tmax": "19,1", "tmin": "3,9", "presMax": "1026,3", "presMin": "1023,1"; "generacion": "12.732" con lo que puede apreciarse que el resultado queda muy cerca del resultado estimado.

Pruebas

Para realizar pruebas, se ha facilitado una opción que relee el fichero de datos climatológicos, realiza una predicción para cada registro, y la escribe en un fichero. De esta forma, podremos extraer los resultados después de entrenar la red neuronal a distintos niveles, y así comparar si hay mejora o no. Puede ser también, que deban utilizarse más hidden layers, aunque en las pruebas realizadas con pocos epochs de entrenamiento no han dado resultados concluyentes al respecto.

Concretamente, las pruebas van a ser realizar entrenamientos de números de iteraciones muy distantes: 50 epochs, 200 epochs o 1000 epochs. Con esto se pretende observar las diferencias entre entrenamientos más o menos exhaustivos.

Veamos pues, los resultados que obtenemos. Van a ser plasmados en un gráfico hecho con **R**, para que pueda compararse visualmente (los resultados pueden encontrarse en los ficheros *.txt* en la carpeta *Resultados*). Como adelanto, decir que durante las ejecuciones (en las que, como se ha podido ver antes, se muestra el error resultante en cada epoch) puede observarse que el error se estabiliza en cierto punto, oscilando muy brevemente (entre 2,9 y 3,1), por lo que es de esperar que las diferencias no serán extremadamente significativas.



(Puede encontrarse el gráfico en el documento *Grafico.png* por si quiere verse con mayor detalle)

Puede apreciarse que para el entrenamiento con 50 epochs las tendencias son más similares. Sin embargo, para 200 y 1000 epochs, las gráficas resultan muy similares entre ellas, pero con cierta distancia con respecto a la original.

Lo que podemos concluir con esto es lo que se ha comentado antes: Los errores varían dentro de un rango, y por tanto, una vez hecha una cantidad mínima de epochs, sabemos que el error se estabilizará.

Es cuestión entonces de probar hasta realizar un *train* que dé como resultado un error bajo, como por ejemplo lo que sucedió con los 50 epochs, acabando con un error de 2,1 (bastante por debajo del rango habitual que oscila entre 3,1 2,9). De esta manera tendremos una red neuronal cuyos resultados serán más fieles.

Como curiosidad recordar la función *trainUntilConvergence()*. Dada la capacidad de entrenamiento de la red, hemos visto que los errores se estabilizan y, aunque tienen picos, vemos que es prácticamente imposible llegar a la convergencia, aunque, si existiera alguna posibilidad, esta función sería la mejor forma de llegar a tal cota de entrenamiento debido a la ingente cantidad de tiempo que sería necesario para que este hecho suceda (la alternativa sería ir comprobando los errores en cada epoch).

Conclusiones

En cuanto a los resultados obtenidos, hemos visto que la oscilación en los diferentes grados de entrenamiento es mínimo. Los valores son verosímiles, por lo que puede descartarse un mal funcionamiento de la red neuronal.

Por tanto, esta oscilación prácticamente homogénea puede deberse a varios factores: el primero podría ser que los valores utilizados de la climatología no son relevantes para la previsión del consumo energético. En este punto, cabe decir que durante las pruebas de entrenamiento se probó eliminando las presiones de los inputs y no se consiguió mejor resultado. No parece que los demás indicadores climatológicas puedan ser de ayuda, lo que nos lleva al siguiente punto: que el consumo eléctrico no esté fuertemente correlacionado con la climatología. Otra opción podría ser que al ser valores de naturaleza tan diferente necesiten una normalización antes de ser tratados.

Cabe destacar que **el número de *hidden layers* seleccionado varía levemente los resultados**. Por defecto en la documentación de *PyBrain* se nos recomienda el uso de 5 *hidden layers*. Sin embargo, donde se han obtenido unos valores más bajos en los errores ha sido aumentando este número hasta los 30 que están activos en el momento de realizar las pruebas anteriores, llegando a ver errores de 1.8 (cosa que no llegabamos a ver con 5 *hidden layers*).

Como conclusiones finales, a modo de sensaciones y aprendizaje, esta ha sido una práctica muy beneficiosa en todos los aspectos. Hemos hecho, durante todas las etapas, cosas nuevas que no habíamos hecho antes en la carrera. Scrapping, uso de un Rest API y, por supuesto hemos usado una red neuronal.

Sin ser una práctica de elevada complejidad (ya se nos advierte de que cojamos una red neuronal hecha), sí tiene un alto componente de aprendizaje, haciéndola sin duda una práctica muy productiva.