Carga de *datasets*

Modelos Avanzados de Aprendizaje Automático I

En esta asignatura se utilizarán distintos modelos avanzados de Aprendizaje Automático para resolver problemas del mundo real. De forma previa al uso de estos modelos, y sin necesidad de avanzar contenidos teóricos, en este ejercicio se propone la carga y preprocesado básico de los *datasets* que se van a utilizar. Estos *datasets* se proveerán en un archivo comprimido, y se pueden dividir en 4 grupos:

- *Datasets* relativos al repositorio PMLB (*Penn Machine Learning Benchmarks*). Este repositorio es una colección de *datasets* estandarizados utilizados para evaluar algoritmos de Aprendizaje Automático. PMLB ofrece una amplia variedad de *datasets* de diferentes dominios y tamaños, que son adecuados tanto para problemas de clasificación como de regresión, con más de 160 *datasets* provenientes de diversas fuentes, lo que permite evaluar algoritmos en una amplia gama de contextos. Además, PMLB está diseñado para integrarse fácilmente con herramientas y bibliotecas como Scikit-Learn, proporcionando una interfaz sencilla para cargar y utilizar los *datasets*. Debido a esto, este repositorio permite realizar evaluaciones sistemáticas y comparaciones entre diferentes algoritmos y configuraciones, promoviendo la investigación reproducible en el campo del Aprendizaje Automático. Para hacer las prácticas de esta asignatura, se proveen 85 *datasets* relacionados con problemas de clasificación en 2 clases. La URL desde donde se puede acceder al repositorio es https://epistasislab.github.io/pmlb/
- Imágenes de uso habitual como *benchmark*. Estas imágenes, como Lena, Cameraman, Peppers, Barbara, Mandrill, reconocidas por sus características específicas y su complejidad visual, han sido fundamentales para el desarrollo y la evaluación de diversas técnicas y algoritmos. Han proporcionado un conjunto de datos consistente para la comparación y validación de métodos de mejora, compresión, restauración y análisis de imágenes, y son ampliamente utilizadas para ilustrar técnicas de filtrado y segmentación, gracias a sus características distintivas y patrones reconocibles. Estas imágenes han jugado un papel crucial en la estandarización de pruebas y han facilitado avances significativos en el campo, sirviendo como referencias comunes en una amplia gama de aplicaciones y estudios académicos. Este conjunto de imágenes puede ser descargado de https://www.imageprocessingplace.com/root_files_V3/image_databases.htm

El dataset MNIST (*Modified National Institute of Standards and Technology*) es uno de los conjuntos de datos más populares y ampliamente utilizados en el campo de la Visión Artificial y el Aprendizaje Automático, especialmente para la tarea de clasificación de imágenes. Este *dataset* se creó a partir de una combinación del conjunto de datos de dígitos escritos a mano del NIST, con la intención de proporcionar un conjunto de datos estándar para la evaluación de algoritmos de reconocimiento de patrones. Las imágenes han sido preprocesadas y normalizadas para centrarlas en una cuadrícula fija de tamaño uniforme (28x28), lo que facilita su uso en una amplia variedad de experimentos y comparaciones de rendimiento entre diferentes métodos. MNIST contiene un total de 70,000 imágenes de dígitos escritos a mano, divididas en un conjunto de entrenamiento de 60,000 imágenes y un conjunto de prueba de 10,000 imágenes. Cada imagen en el *dataset* tiene un tamaño de 28x28 píxeles y está en escala de grises, representando dígitos del 0 al 9.

MNIST ha sido fundamental en el desarrollo y la validación de algoritmos de aprendizaje automático y redes neuronales, sirviendo como un punto de partida accesible y sencillo para la investigación en clasificación de imágenes. Su simplicidad y claridad lo hacen ideal para los principiantes en el campo, mientras que su amplia aceptación y uso en la comunidad de investigación permiten comparaciones directas y *benchmarking* de nuevos algoritmos y técnicas. Gracias a su robustez y facilidad de uso, MNIST ha contribuido significativamente a los avances en el reconocimiento de patrones y ha inspirado la creación de muchos otros conjuntos de datos más complejos en el ámbito de la Visión Artificial.

El *dataset* original puede ser descargado en http://yann.lecun.com/exdb/mnist/

Sin embargo, en esta asignatura se provee de un archivo en formato jld2 con todo el *dataset*. Este se carga como una variable de tipo *Dict*, con 4 entradas:

- "train_imgs", de tipo Vector{Matrix{Float32}}, es decir, es un vector de 60000 valores, donde cada uno es una matriz de tamaño 28x28 con cada imagen en escala de grises, con valores entre 0 y 1.
- "train_labels", de tipo Vector{Int64}, con 60000 valores entre 0 y 9.
- "test_imgs", de tipo Vector{Matrix{Float32}}, es decir, es un vector de 10000 valores, donde cada uno es una matriz de tamaño 28x28 con cada imagen en escala de grises, con valores entre 0 y 1.
- "test_labels", de tipo Vector{Int64}, con 10000 valores entre 0 y 9.

- *Dataset* relativo al tipo de aprendizaje denominado *Stream Learning* o aprendizaje en flujo de datos. Concretamente, el *dataset* a utilizar se llama *Electricity*, y es uno de los conjuntos de datos más utilizados en el campo del aprendizaje en flujo y el aprendizaje incremental. Este dataset se originó a partir del mercado de la electricidad en Nueva Gales del Sur, Australia, y

contiene datos horarios de precios de la electricidad y demanda, así como otros factores relacionados, en un mercado competitivo. Cada registro en el conjunto de datos representa la situación en un punto en el tiempo específico y se utiliza para predecir si el precio de la electricidad en una determinada hora será más alto o más bajo que el precio de la hora anterior. Este *dataset* se puede descargar de la siguiente dirección: https://github.com/vlosing/driftDatasets/tree/master/realWorld/Elec2

Las variables independientes de este *dataset* son las siguientes:

- *date*: Fecha y hora del registro.
- *day*: Día de la semana (números naturales entre 1 y 7)
- *period*: Periodo del día (1-48, para intervalos de media hora).
- *nswprice*: Precio de la electricidad en Nueva Gales del Sur.
- *nswdemand*: Demanda de electricidad en Nueva Gales del Sur.
- *vicprice*: Precio de la electricidad en Victoria.
- *vicdemand*: Demanda de electricidad en Victoria.
- *transfer*: Cantidad de electricidad transferida entre Nueva Gales del Sur y Victoria.

En el artículo original, se recomienda eliminar los atributos 1 (*date*) y 4 (*nswprice*). El resto de atributos, excepto *day*, están normalizados. Este atributo *day* tiene valores cíclicos que tienen que ser codificados como se muestra más adelante.

Por su parte, la variable objetivo, *class*, indica si el precio de la electricidad es más alto (UP) o más bajo (DOWN) en comparación con la hora anterior. Este *dataset* se provee como dos archivos:

- elec2_data.dat, con los atributos como una matriz de tamaño 45312x8.
- elec2_label.dat, con las etiquetas, como una matriz de tamaño 45312×1.

Para cargar estos datasets, es necesario antes importar (*using*) los siguientes paquetes en Julia: *FileIO*, *Images*, *JLD2*.

Como se muestra, el *dataset* relativo a *Stream Learning* contiene un atributo con valores cíclicos, en este caso relativos a días de la semana, codificados como números naturales del 1 al 7. En un caso de tener pocos valores posibles, como es el caso, una opción habitual es realizar un *one-hot-enconding*. Sin embargo, otras veces el número de valores distintos puede ser mucho mayor (como el número de días en un año), o incluso ser un valor continuo, por lo que realizar la codificación *one-hot-enconding* no resulta práctica. En este ejercicio, se realizará la codificación habitual cuando el número de valores cíclicos es alto, o bien son valores continuos. Para ello, es necesario distinguir dos casos distintos, según si los valores cíclicos son discretos o continuos. Un ejemplo del primer caso, con valores discretos, está en el atributo a preprocesar en este *dataset*: los días de la semana. Otros

ejemplos típicos podrían ser las estaciones, los días del año, o las horas del día tomando únicamente la hora, sin fracción de minutos/segundos. Un ejemplo del segundo caso, con valores continuos, podría ser la hora del día, pero tomando la fracción de minutos/segundos. En general, cualquier atributo cíclico en el que se tenga un número muy alto de valores podría ser considerado como continuo. La diferencia entre ambos tipos es que, en el caso de valores discretos, el último valor no equivale al primero, y en el caso de valores continuos esto sí que pasaría, por lo que esta situación tiene que ser considerada dentro de la formulación. Por ejemplo, en los días de la semana el día 7 no equivale al día 1; sin embargo, en la hora del día, el instante 23:59:59.99 sí equivale al instante 00:00:00 del día siguiente (a no ser que se necesite una precisión extremadamente alta).

En general, para realizar la codificación, se hace primero una normalización entre máximo y mínimo de los valores, puesto que están acotados, pasando a un intervalo no entre 0 y 1, sino entre 0 y 2π, y después se construyen dos nuevos atributos, con los senos y los cosenos de estos ángulos. En el caso de atributos continuos, la circunferencia se rellenará casi en su totalidad, mientras que en el caso de los atributos discretos se tomarán tantos ángulos equiespaciados como valores se tengan, siendo el último valor no igual a 2π (esto conllevaría que el último sea igual al primero).

Para hallar estos valores de ángulos, para el caso continuo, la normalización y transformación en ángulo se puede hacer mediante la ecuación

$$rad = 2\pi \frac{x - min}{max - min}$$

donde x es el valor del atributo. En esta ecuación, la parte $\frac{x-min}{max-min}$ transforma x para pasarlo al intervalo [0,1], y, al multiplicarlo por 2π se convierte en valores entre 0 y 2π.

Sin embargo, como se ha dicho, para valores discretos esto no sería correcto, puesto que con esta ecuación el valor máximo coincidiría con el ángulo 2π, que es igual al ángulo 0, es decir, el primer valor. Para ello, una solución sencilla es aumentar el intervalo en el que están los valores. Por ejemplo, los días de la semana, en lugar de estar entre 1 y 7, se considera que el valor máximo es el 8. De esta forma, el valor 7 no ocupará el ángulo 0, que tiene el primer valor. En general, para hacer esto es necesario calcular la magnitud de cambio entre valores contiguos de este atributo. Para los días de la semana sería un valor de 1, pero este valor podría ser distinto, por ejemplo, si las 4 estaciones del año se tienen como [0, 0.25, 0.5, 0.75], este valor sería igual a 0.25. Una vez que se tiene este valor, la ecuación sería muy similar a la anterior:

$$rad = 2\pi \frac{x - min}{max - min + m}$$

donde m sería este valor de cambio. Estas dos ecuaciones para estos dos casos (continuo y discreto) se pueden unir en una sola, igual a la anterior, teniendo en cuenta que m=0 para el caso continuo.

En lo que respecta al procesado de imágenes, a la hora de trabajar con varias imágenes, es habitual unir estas imágenes en una única matriz de 4 dimensiones, donde 2 de las dimensiones son altura (H) y ancho (W), otra es el número de canales (C, 3 para imágenes en color RGB y 1 para imágenes en escala de grises), y otra dimensión es el número de imágenes (N). De esta forma, se tiene una única variable con una matriz de 4 dimensiones. Según el orden de estas dimensiones, esta puede ser HWCN, NCHW o NHWC. En esta práctica, el formato a utilizar será NCHW, es decir:

- N: número de imágenes.
- C: número de canales. Como en esta práctica sólo se trabajará con imágenes en escala de grises, esta dimensión sólo tendrá un canal.
- H: altura de la imagen.
- W: anchura de la imagen.

De esta manera, por ejemplo, si se tiene las imágenes en una variable en formato NCHW llamada *images* y se quiere referenciar la imagen 27, se puede hacer fácilmente con:

```
images[27,1,:,:]
```

En este ejercicio, se dan las siguientes funciones hechas:

- *convertImagesNCHW*. Recibe un vector de matrices, donde cada una es una imagen en escala de grises, y devuelve una variable con todas las imágenes en formato NCHW
- *showImage*. Esta función permite mostrar imágenes en pantalla. Tiene 3 versiones o métodos:
    - Recibe una matriz (bidimensional) y la muestra en pantalla.
    - Recibe una variable NCHW y muestra todas las imágenes, una al lado de otra.
    - Recibe dos variables NCHW y muestra todas las imágenes, una al lado de otra, con las de la primer variable encima de las de la segunda variable.
- *intervalDiscreteVector*. Esta función permite calcular el valor de m de la ecuación anterior relativa a la codificación de datos cíclicos. Recibe un vector de valores reales, e intenta calcular el valor de cambio entre los valores. Si son valores discretos, devuelve este valor de cambio. Si son valores continuos, devuelve un valor de 0. Con esta función, las dos ecuaciones anteriores para codificar valores cíclicos, continuos y discretos, se pueden unir en una sola de forma muy sencilla.

En este ejercicio, se pide realizar las siguientes funciones:

- *fileNamesFolder*. Devuelve los nombres de archivos que hay en una carpeta. Recibe:
    - *folderName*, de tipo *String*, con el nombre de una carpeta a examinar.
    - *extension*, de tipo *String*, con la extensión que tienen los nombres de archivos a buscar, sin el punto.

  Esta función debe devolver un vector de valores de tipo *String*, con los nombres de los archivos encontrados en esa carpeta, pero con los nombres sin la extensión.

  Para desarrollar esta función puede ser útil utilizar las funciones *isdir*, *readdir*, *uppercase*, *filter*. Una forma sencilla de tomar los nombres de archivo de una carpeta que tengan una determinada extensión puede ser haciendo lo siguiente:

```
extension = uppercase(extension);
fileNames = filter(f -> endswith(uppercase(f), ".$extension"), readdir(folderName))
```

  De esta forma, sólo sería necesario eliminar la extensión de los nombres de los archivos.

  En el desarrollo de esta función no se permite el uso de bucles.

- *loadDataset*. A partir del nombre del *dataset* y de la carpeta donde está, carga el *dataset* correspondiente y lo devuelve. Recibe como parámetros obligatorios:
    - *datasetName*, de tipo *String*, con el nombre del *dataset* sin la extensión. En general, en los *datasets* tomados del repositorio PMLB, el nombre del *dataset* será el nombre del archivo sin la extensión. Por tanto, el primer paso será añadir la extensión ".tsv" al nombre del archivo.
    - *datasetFolder*, de tipo *String*, con el nombre de la carpeta donde está el *dataset*.

  Como parámetro opcional, tiene:

    - *datasetType*, de tipo *DataType*, con el tipo de dato a cargar para las entradas. Por defecto tiene como valor *Float32*, que es el tipo comúnmente utilizado con Redes de Neuronas.

  Esta función cargará el *dataset* correspondiente y lo devolverá como una tupla (*inputs*, *targets*), donde inputs será una matriz de valores del tipo especificado en *datasetType* y

targets será un vector de valores booleanos con la salida deseada (todos los *datasets* que se proveen son de 2 clases). Si no se encuentra el archivo en cuestión, esta función devolverá *nothing*.

Algunas funciones que pueden ser útiles para desarrollar esta función son *joinpath*, *abspath*, *readdlm* (esta está dentro del paquete *DelimitedFiles*), *findall*, *unique*, *setdiff*.

Para cargar el archivo, se puede usar la función *readdlm*, especificando el tabulador como separador con '\t'. Esto devolverá una matriz, donde la primera fila serán los encabezados. Lo primer que hay que hacer es hallar qué columna se corresponde con la salida deseada, será aquella cuyo encabezado sea igual a "target". Una vez hallada esta columna, se toman las entradas y las salidas deseadas de la siguiente forma:

- Entradas: filas de la segunda a la última, y todas las columnas excluyendo la de las salidas deseadas. Será necesario convertir estos valores al tipo especificado en la llamada.
- Salidas deseadas. En los *datasets* utilizados las salidas deseadas serán valores en el conjunto {0,1}, por lo que se pueden convertir fácilmente a valores booleanos.

Para el desarrollo de esta función no se permite el uso de bucles.

- *loadImage*. Dado el nombre de una imagen (sin la extensión) y el nombre de la carpeta, carga la imagen y la devuelve como una matriz bidimensional. Esta función recibe como parámetros obligatorios:

  - *imageName*, de tipo *String*, con el nombre de la imagen sin la extensión. Todas las imágenes a utilizar (excluyendo el *dataset* MNIST) tienen extensión ".tif"

  - *datasetFolder*, de tipo *String*, con el nombre de la carpeta donde se buscará el archivo de imagen.

  Como parámetros opcionales, tiene:

  - *datasetType*, de tipo *DataType*, con el tipo de datos que tendrá que tener la matriz con la imagen cargada.

  - *resolution*, de tipo *Int*, con la resolución de la imagen. Solamente es un valor, porque todas las imágenes serán cuadradas. Cambiar la resolución de una imagen se puede hacer con la función *imresize*, del paquete *Images*, con una llamada similar a:

```
image = imresize(image, (resolution,resolution));
```

Funciones que pueden ser útiles: *abspath*, *joinpath*, *isfile*, *load*, *imresize*, *gray*.

El primer paso será añadir la extensión al nombre del archivo, y posteriormente cargar la imagen (*load*), modificar su resolución (*imresize*), convertirla a una matriz (*gray*), y cambiar el tipo de la matriz. Si el archivo no existe, esta función devolverá *nothing*.

Para el desarrollo de esta función no se permite el uso de bucles.

- *loadImagesNCHW*. Carga todas las imágenes que haya en una carpeta y las devuelve en formato NCHW. Recibe como parámetros obligatorios:
    - *datasetFolder*, de tipo *String*, con la carpeta donde están las imágenes.

  Como parámetros opcionales, tiene:
    - *datasetType*, de tipo *DataType*, con el tipo de datos.
    - *resolution*, de tipo *Int*, con la resolución de las imágenes.

  Esta función se puede hacer mediante una llamada a la función *fileNamesFolder*, indicando la extensión "tif", una llamada haciendo un *broadcast* a *loadImage* con el resultado de la llamada anterior, y, posteriormente, una llamada a *convertImagesNCHW* con el resultado de esta llamada anterior.
  Para el desarrollo de esta función no se permite el uso de bucles.

- *loadMNISTDataset*. Carga el dataset de MNIST y lo devuelve como 4 variables, con las entradas y salidas deseadas en entrenamiento y test. Recibe como parámetro obligatorio:
    - *datasetFolder*, de tipo *String*, con el nombre de la carpeta donde está el archivo "MNIST.jld2".

  Como parámetros opcionales, tiene:
    - *labels*, de tipo *AbstractArray{Int,1}*, con las etiquetas a cargar.
    - *datasetType*, de tipo *DataType*, con el tipo de datos a devolver.

  En la llamada a esta función se especifica, de forma opcional, qué etiquetas se quieren cargar. La base de datos MNIST tiene todos los dígitos, del 0 al 9, pero es posible que el

usuario desee hacer experimentos solamente con unos pocos dígitos. Por ejemplo, si sólo se quiere trabajar con los dígitos 2, 5 y 8, el parámetro *labels* será igual a [2,5,8]. De forma adicional, si *labels* tiene algún valor igual a -1, esto quiere decir que el resto de imágenes con etiquetas no contempladas en *labels* serán reunidas en la etiqueta -1. Esto es útil para hacer estrategias similares a "uno contra todos". A continuación se muestran 2 ejemplos de uso:

- Si *labels*=[3,7], el conjunto de imágenes será muy inferior al conjunto cargado, puesto que sólo tendrá las imágenes correspondientes a los dígitos 3 y 7.
- Si *labels*=[3,7,-1], el conjunto de imágenes será igual al conjunto cargado, con todas las imágenes. Sin embargo, las imágenes que no sean ni 3 ni 7 tendrán una etiqueta igual a -1.

Para cambiar los valores de las salidas deseadas (*targets*) que no estén dentro de *labels* a -1, se puede hacer con

```
targets[.!in.(targets, [setdiff(labels,-1)])] .= -1;
```

Para saber cuáles son las imágenes a tomar por tener sus salidas deseadas dentro de *labels*:

```
indices = in.(targets, [labels])
```

Posteriormente, es necesario hacer llamadas a *convertImagesNCHW* para convertir los vectores de imágenes cargadas a formato NCHW.

Esta función debe devolver una tupla con 4 elementos, por este orden:

- Matriz de imágenes de entrenamiento en formato NCHW en el formato especificado.
- Vector de valores enteros con las salidas deseadas de entrenamiento, tomando las etiquetas especificadas.
- Matriz de imágenes de test en formato NCHW en el formato especificado.
- Vector de valores enteros con las salidas deseadas de test, tomando las etiquetas especificadas.

Para el desarrollo de esta función no se permite el uso de bucles.

- *cyclicalEncoding*. Dado un vector de valores reales, interpreta estos datos como cíclicos, y realiza su codificación como senos y cosenos. Recibe como parámetro obligatorio:

  - *data*, de tipo *AbstractArray{<:Real,1}*, con los valores del atributo a codificar.

  Funciones que pueden ser útiles: *maximum*, *minimum*, *extrema*, *intervalDiscreteVector* (esta se provee en este ejercicio), *sin*, *cos*. Esta función se apoya en una llamada a la función *intervalDiscreteVector*, que devuelve el valor de m de la ecuación anterior, que puede tener

un valor igual a 0, para aplicar la ecuación anterior. Una vez aplicada, solamente queda realizar el cálculo de los senos y los cosenos de estos ángulos.

Esta función debe devolver una tupla con 2 vectores: senos y cosenos (por este orden) resultados de realizar esta codificación.

Para el desarrollo de esta función no se permite el uso de bucles.

- *loadStreamLearningDataset*. Devuelve el *dataset* utilizado para resolver el problema de *Stream Learning*. Recibe como parámetro obligatorio:

    - *datasetFolder*, de tipo *String*, con la carpeta donde están los archivos "elec2_data.dat" y "elec2_label.dat"

  Como parámetro opcional, tiene:

    - *datasetType*, de tipo *DataType*, con el tipo de dato a devolver para las entradas.

  Funciones que pueden ser útiles: *joinpath*, *abspath*, *readdlm*, *vec*, *setdiff*, *cyclicalEncoding* (esta se desarrolla en este ejercicio).

  Este *dataset* se provee como dos archivos, de nombres "elec2_data.dat" y "elec2_label.dat", con las entradas y salidas deseadas respectivamente. Ambos archivos se pueden cargar con la función *readdlm*. Las salidas deseadas convertirán a booleanos, y posteriormente a vector con *vec*. En cambio, en las entradas hay que hacer más preprocesado:
    - Eliminar las columnas 1 (*date*) y 4 (*nswprice*).
    - Llamar a *cyclicalEncoding* con los valores de la primera columna (día) resultante de la operación anterior.
    - Concatenar las salidas de senos y cosenos, por este orden, a la matriz resultante de eliminar la primera columna con el día, es decir, ponerlas como las primeras columnas de la matriz de entradas a devolver.

  Esta función deberá devolver una tupla con 2 elementos, por este orden:
    - Entradas, como una matriz bidimensional con valores del tipo especificado. Esta matriz tendrá 7 columnas, con los valores, por este orden, de senos, cosenos, *period*, *nswdemand*, *vicprice*, *vicdemand* y *transfer*.
    - Salidas deseadas, como un vector de valores booleanos.

  Para el desarrollo de esta función no se permite el uso de bucles.

Firmas de las funciones:

A continuación se muestran las firmas de las funciones a realizar en los ejercicios propuestos. Tened en cuenta que, dependiendo de cómo se defina la función, esta puede contener o no la palabra reservada *function* al principio:

```
function fileNamesFolder(folderName::String, extension::String)
function loadDataset(datasetName::String, datasetFolder::String; datasetType::DataType=Float32)
function loadImage(imageName::String, datasetFolder::String; datasetType::DataType=Float32,
       resolution::Int=128)
function loadImagesNCHW(datasetFolder::String; datasetType::DataType=Float32, resolution::Int=128)
function loadMNISTDataset(datasetFolder::String; labels::AbstractArray{Int,1}=0:9,
       datasetType::DataType=Float32)
function cyclicalEncoding(data::AbstractArray{<:Real,1})
function loadStreamLearningDataset(datasetFolder::String; datasetType::DataType=Float32)
```