Métricas para evaluación de modelos. Introducción a imágenes digitales.

Práctica 3

Ejercicio 1. En el archivo "Práctica 3 - Métricas de clasificación.ipynb" encontrarás un notebook donde deberá implementar varias de las métricas vistas en clase.

Ejercicio 2. Entrená un modelo que clasifique correctamente las clases en el archivo "6_clases_dificil.csv". Realizá un análisis de las diferentes métricas vistas en clase. Podés utilizar las funciones definidas en *AAPutils.py* para realizar más rápidamente los gráficos y reportes. A continuación se muestra un breve glosario de algunas funciones que te serán de utilidad.

AAPutils.py	
X_train, X_test, Y_train, Y_test = **dividir_train_test**(X, Y, test_size)	Divide el dataset de forma aleatoria en dos conjuntos. El parámetro "test_ size" es un número entre 0 y 1 que indica el porcentaje a utilizar para el conjunto de evaluación.
plot_frontera_de_decision_2D(model, X_train, Y_train, X_test, Y_test)	Grafica en 2D el dataset junto con la frontera de decisión del modelo. Al pasar el conjunto de testeo se visualizan los dos conjuntos con diferentes marcadores.
plot_training_curves(history , acc=True)	Grafica las curvas de entrenamiento (loss) para el training y testing set. Si el parámetro "acc" es True grafica también la curva para el Accuracy.
plot_confusion_matrix(y_true, y_pred)	Grafica la matriz de confusión para los vectores de clases pasados como argumento.
print_classification_report(y_true, y_pred)	Imprime en consola un reporte del accuracy para los vectores de clases pasados como argumento. Si la clasificación es binaria se imprime además el precision, recall y f-measure.
plot_ROC_curve(modelo, x, y)	Grafica las curvas ROC y precision-recall para el modelo entrenado y para el conjunto pasado como argumento.

Ejercicio 3. Realizá el mismo procedimiento que el ejercicio 2, pero para los datasets descriptos a continuación. Recorda que las redes neuronales son sensibles a la escala de los datos. Intentá encontrar un balance adecuado entre **precision** y **recall**.

Titanic.csv	Información sobre diferentes pasajeros del Titanic indicando si sobrevivió o no al viaje.
Diabetes.csv	Dataset con información sobre pacientes, indicando si cada uno tiene o no diabetes gestacional.
Ecoli.csv	Información de proteínas de pacientes, indicando si presenta o no una enfermedad.

En los conjuntos de Diabetes y Ecoli, ¿cuál es el **precision** más alto que podés obtener, manteniendo el **recall** cercano a 1?

Ejercicio 4. Realizá el tutorial sobre imágenes digitales que encontrarás en "Practica 3 - Procesamiento de imágenes.ipynb".

Ejercicio 5. MNIST es una base de datos que contiene 70 mil imágenes en escala de grises de dígitos escritos a mano. Genere un modelo que permita clasificar las 10 clases.

a) Carga de datos

El código para descargar el dataset ya se encuentra en Keras y es posible cargarlo en memoria del siguiente modo:

```
from keras.datasets import mnist
(X_train, Y_train), (X_test, Y_test) = mnist.load_data()
```

Los arreglos X_train y X_test poseen las imágenes del dataset y cada uno tiene un shape: (N, H,W), donde N es la cantidad de ejemplos, H el alto de la imagen y W el ancho. En el caso de MNIST tanto H como W valen 28.

b) Visualización

El primer paso para cerciorarse de que un conjunto de datos de imágenes está cargado correctamente es visualizar las mismas.

Para visualizar las imágenes, podés utilizar la función plt.imshow(). Por ejemplo, para ver la primera imagen del conjunto de entrenamiento, podemos ejecutar:

```
plt.imshow(X_train[0, :,:])
```

Para ver la tercera imagen del conjunto de prueba:

```
plt.imshow(X_test[2, :,:])
```

Implementá un código que visualice 8 imágenes de cada clase en una grilla de 8x10. Utiliza la función **subplots** de **matplotlib.pyplot** para generar la grilla.

c) Aplanamiento

Para clasificar con una red neuronal con capas del tipo **x*W+b** (clase Dense en Keras), el formato de imagen (N,H,W) no sirve. Por ende los ejemplos deben adaptarse al formato (N,V), donde V es la cantidad de variables, que en este caso sería V=H*W=28*28= 784. La capa **Flatten** realiza esta operación:

```
model.add(Flatten(input_shape=(28,28)))
```

La operación **Flatten** se refiere a **aplanar** un arreglo n-dimensional, es decir, desarmar las estructuras de sus dimensiones y quedarse con todos los valores, pero ahora en un vector de una sola dimensión. En este caso, solo se desarman las dimensiones 1 y 2, la dimensión 0 queda porque es la que separa los ejemplos. Como no se pierde ningún valor, sólo se reordenan, la nueva dimensión V tiene tamaño 784=H*V=28*28.

*Nota: La API de Keras es inconsistente, y utiliza el parámetro **input_shape** en la clase Flatten, mientras que en el resto de capas se utiliza el parámetro **input_dim**. Recordamos que este parámetro sólo es necesario en la primera capa para especificar el tamaño de la entrada.*

d) Entrenamiento del modelo

Entrená un modelo para clasificar las imágenes, utilizando una softmax en la capa de salida y la entropía cruzada como función de error. Medí el error y el accuracy en el conjunto de test (y el de train). Computá la matriz de confusión, pero antes de mirarla ¿qué pares de clases te parece que van a confundirse más?.

e) Normalización de las imágenes

Los pixeles de las imágenes están codificadas en el rango 0-255. Es más beneficioso para el entrenamiento de la red que estén normalizadas, con media 0 y varianza 1. La normalización debe realizarse a nivel de pixel, es decir, se debe calcular la media de todos los pixeles de todas las imágenes, luego la varianza de todos los pixeles de todas las imagenes y luego realizar la normalización. En este caso tanto la media como la varianza son escalares, ya que cada pixel está codificado por un solo valor. Implementa dicha normalización utilizando numpy, antes de las etapas de entrenamiento y clasificación.

Ejercicio 6. Realice el mismo procedimiento que el ejercicio 5, pero para el *dataset* **CIFAR10**. Este conjunto de datos posee imágenes de 10 clases de la vida cotidiana como diferentes animales y vehículos.

a) Carga de datos

```
from keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

b) Visualización

Para visualizar las imágenes, también podés utilizar la función plt.imshow(), pero ahora tené en cuenta la dimensión extra. Modificá el código:

```
plt.imshow(x_train[0, :,:,:]) #incluir la dimensión de canales o colores
```

c) Aplanamiento

Las imágenes de CIFAR10 son a color, por ende las dimensiones de **x_train** y **x_test** son del tipo (N,H,W,C), donde C es la cantidad de canales (3 para imágenes a color). En el caso de CIFAR10, las imágenes son de tamaño 32x32, por ende el **shape** de estos vectores es (N,32,32,3). En este caso también tenemos que utilizar la capa Flatten al principio del modelo, pero ahora con el nuevo tamaño de entrada:

```
model.add(Flatten(input_shape=(32,32,3)))
```

d) Entrenamiento del modelo

Entrená un modelo para clasificar las imágenes, utilizando una softmax en la capa de salida y la entropía cruzada como función de error. Medí el error y el accuracy en el conjunto de test (y el de train). Computá la matriz de confusión, pero antes de mirarla ¿qué pares de clases te parece que van a confundirse más?

e) Normalización

La normalización para el caso de CIFAR10 se debe realizar igual que con MNIST, pero ahora como hay 3 canales (uno por cada color) la normalización debe realizarse por cada canal.