ML workflow

Franco Sebastián Torres

March 2025

1 Introduction

1.1 Carga de datos

En primer lugar se tienen los datos crudos y se proceden a cargar para su posterior preprocesamiento (en caso de que lo requieran). Los datos cargados en esta instancia seran usados tanto para entrenamiento y validación del modelo, como para testeo. En este caso los datos utilizados son muestras de audio.

2 Creación del impulso

En el contexto de *EdgeImpulse* (EI), un impulso es un flujo de procesamiento que sigue el modelo de ML. Es el conjunto de pasos que transforma los datos de entrada en predicciones útiles (en un modelo que haga predicciones :)). Basicamente representa el workflow... Un impulso se divide en 3 bloques principales:

- Input block: Es donde se cargan los datos. Este bloque puede realizar un ventaneo de datos.
- Processing block: Extrae los features de los datos. [Explicar que es un feature en la intro]
- Learning block: Es la parte donde se crea el modelo y se entrena con los features obtenidos.

En las siguientes secciones vamos a ver cada uno de los bloques.

2.1 Input block

En este bloque se cargan principalmente los datos para usar en el modelo y se tienen opciones para obtener mas features. En EI este bloque marca los *input axes* que es basicamente los distintos ejes o dimensiones de los datos de entrada. Luego se tiene *Windows size* que es el tamaño del segmento de datos crudos que se va a usar para entrenar el modelo. Luego sigue *Windows incrase* que basicamente es el desplazamiento de la ventana, todo esto con el fin de crear mas datos de entrenamiento sin tener que recolectar mas datos.

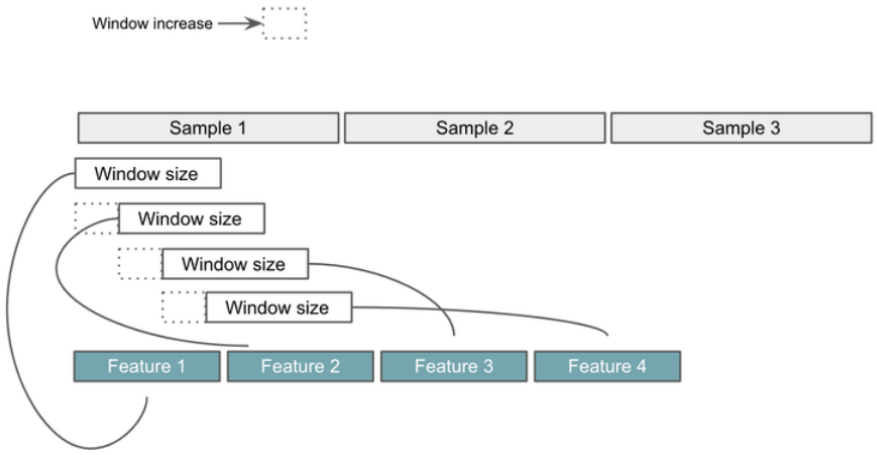

Figure 1: En este caso los features que aparecen en el esquema no son los que vamos a usar para entrenar el modelo sino que hay que hacerle un procesamiento previo en el proximo bloque. Vamos a llamarle ventanas simplemente.

Frequency se calcula automaticamente basada en los datos de entrada pero se puede modificar. Finalmente se tiene la opción de *Zero-pad data*, basicamente lo que hace es completar con ceros en los casos donde falten datos para completar una ventana.

En nuestro caso los datos utilizados son audios de un solo canal por lo que se tiene un solo eje. Luego la frecuencia de los audios varia segun el dispositivo usado en la recolección.

[La idea seria encontrar la implementación de un código o hacer uno, lo cual en ppio no parece tan complicado.]

2.2 Processing block

Se encarga de extraer caracterisitcas (features) significativas a partir de los datos de entrada (las ventanas en este caso). Hay varios tipos de bloques de procesamiento por lo que solo vamos a ver algunos.

MFCC

Mel frequency crepstal coefficients: Son coeficientes que capturan información relevante sobre el audio. Se utilizan principalmente para el reconocimiento de voz, pero también pueden aplicarse a otros sonidos. El proceso es el siguiente:

- Se separa la señal en ventanas.
- Se calcula la FFT.

- Se aplica un banco de filtros en la escala de Mel. (y se calcula la energia $E_m = \sum_{k=0}^{N-1} x(k)^2 H_m(k), 1 \leq m \leq F$ Con F el nro de filtro y N la FFt length.
- Se toma el logaritmo del resultado y se aplica la Transformada Discreta del Coseno (DCT).
- Se obtienen los coeficientes, de los cuales los primeros son los más importantes para voz.
- Se repite este proceso para cada ventana.

Mel-filterbank: Es un conjunto de filtros triangulares equiespaciados en la escala de Mel. Estos filtros están diseñados para imitar la percepción humana del sonido, dándole mayor importancia a las frecuencias bajas. La conversión entre frecuencia en Hz y la escala de Mel se realiza mediante la ecuación:

$$f_{mel} = 2595 \log_{10} \left(1 + \frac{f_{Hz}}{700}\right) \tag{1}$$

El cálculo de los filtros en la escala de Mel se puede hacer con el siguiente código:

```
import numpy as np
sr=32000 # Frecuencia de muestreo
fftLength=4096 # Largo de la FFT
mel_num=512 # Número de filtros Mel
fmin=0
fmax=16000
freArr=np.linspace(0, sr/2, fftLength/2+1)

# Conversión de Hz a escala Mel
fmin_mel=2595*np.log10(1+ fmin/700)
fmax_mel=2595*np.log10(1+ fmax/700)
freArr_scale=np.linspace(fmin_mel,fmax_mel,mel_num)
freArr_mel=700*(np.power(10, freArr_scale/2595)-1)
```

Luego de aplicar estos filtros, se obtiene una representación comprimida de la información del espectro. Posteriormente, se aplica un logaritmo y una DCT para reducir la redundancia y extraer los coeficientes MFCC finales.

En EI tengo un par de parámetros en esta sección:

- Número de coeficientes que se conservan luego de aplicar DCT (gralmente se usan 12-13 para conservar toda la info relevante)
- *Frame length*: Es el tamaño de la ventana que se usa en la MFCC (supuestamente una ventana grande puede mezclar sonidos)
- *Frame stride*: Es el desplazamiento de ventana.

- Número de filtros para usar en la escala MEL.
- *FFT length* (N_{FFT}) Debe ser potencia de 2 y define la resolución de la frecuencia como $\Delta f = \frac{sr}{N_{FFT}}$. Para tamaños pequeños tengo buena resolución temporal pero pobre en frecuencia y para tamaños grandes tengo poca resolución temporal pero mejoro la de frecuencia.
- Low-High frequency: definen el rango de frecuencias que se van a usar para el análisis.
- *Normalization windows size*: se usa para eliminar "ruido" de los coeficientes producto de variaciones rápidas, por eso se le resta la media de los coef que estan al rededor [Buscar info y completar]

Por lo que la salida de este bloque para cada ventana del segmento anterior va a ser una matriz. Para cada ventana de input se hace otro ventaneado para calcular los mfcc y obtener 13 coeficientes.

Supongamos que le doy un audio de 100ms y mi ventaneado es de 20ms con stride de 20ms, entonces tengo 5 frames donde por cada uno obtengo 13 coeficientes entonces tengo $13 \times 5 = 65$ valores (o una matriz de 5×13) o un array unidimensional de 65 valores.

2.3 Learning block

Classification with keras

En este bloque se crea la red neuronal y se la entrena. La idea principal es se tiene una red neuronal que toma un input y devuelve las distintas probabilidades de pertenecer a alguna de las clases.

El modelo sigue la siguiente estructura, donde B es el tamaño del batch y $N = \frac{\text{input\_length}}{13}$ representa la cantidad de ventanas de 20ms en un segundo de audio.

Capa	Output shape
Entrada	$(B, \text{input\_length})$
Gaussian Noise (stddev=0.45)	$(B, \text{input\_length})$
Reshape	$(B, N, 13)$
Conv1D (8 filtros, kernel=3, padding='same', ReLU)	$(B, N, 8)$
MaxPooling1D (pool=2, stride=2, padding='same')	$(B, N/2, 8)$
Dropout (0.25)	$(B, N/2, 8)$
Conv1D (16 filtros, kernel=3, padding='same', ReLU)	$(B, N/2, 16)$
MaxPooling1D (pool=2, stride=2, padding='same')	$(B, N/4, 16)$
Dropout (0.25)	$(B, N/4, 16)$
Flatten	$(B, N/4 \times 16)$
Dense (classes, activación='softmax')	$(B, \text{classes})$

Vamos a ver paso por paso cada una.

2.3.1 Reshape

En primer lugar se tiene la capa de entrada donde se mandan N ventanas de 13 (según el numero que pongamos) MFCC cada una. A ese array unidimensional se le agrega ruido gaussiano (si se activo en data augmentation) y eso se manda a la siguiente capa.

La capa de Reshape recibe un vector de forma $(N*13)$ y lo transforma en una matriz de forma $(N, 13)$ donde cada fila representa una ventana y cada columna un MFCC. Este paso es importante porque la capa Conv1D espera recibir un tensor de la forma (B, x, y) donde x representa el numero de ventanas e y el nro de canales por ventana.

2.3.2 Conv1D

La capa Conv1D(fn,kz,pd,af) es convolucional, es decir toma un kernel de elementos en el dominio temporal (filas) y los suma (sobre todas las columnas) aplicando un peso. El tamaño de kernel (kz) define cuantas filas participan en la suma. El stride marca el paso (cuantas filas nos movemos hacia abajo antes de aplicar nuevamente el filtro.), en este caso es 1 (default). Padding se refiere a como se tratan los datos si al movernos no se llega a completar la dimensión de kernel ("same" hace que la cantidad de veces que se aplica el filtro sea igual al nro de filas). Luego por cada filtro (definido por fn) se tienen pesos distintos y un canal mas de salida.

El tema de los pesos: los pesos ahora corresponden al filtro, no a la conexión entonces tengo W= $(13 \times kz + 1) \times fn$, donde el 1 corresponde al termino de bias que se suma al total.

Finalmente se tiene una función de activación que ya sabemos lo que hace. ReLU(x) = max(0,x).

Como se menciono antes, al tener 8 filtros y padding "same" la salida es de la forma $(B, N, 8)$.

2.3.3 MaxPooling1D

Siguiente sigue la capa de MaxPooling1D (pool,stride,padding) que lo que hace es tomar una ventana de elementos consecutiva (pool) y su output para ese caso es un elemento que corresponde al máximo de los del pool. En este caso cada columna es independiente y los elementos se toman recorriendo las filas, entonces el output tendría una forma $(B, N/pool, 8)$ donde $N/pool$ se redondea hacia arriba por tener el padding en same.

Supuestamente es importante porque reduce el tamaño de la salida -¿ menos costo y al conservar el valor máximo (asumiendo que es lo mas importante) nos quedamos con las características relevantes.

2.3.4 Dropout

Luego sigue una capa de Dropout(dp) donde dp define el ratio de dropout. Básicamente es un filtro con el tamaño de mi entrada donde aleatoriamente un

ratio dp de los elementos son fijados a cero. El resto se los divide por $1 - dp$ con el fin de que el valor de expectación se mantenga constante. Sirva bastante para evitar overfitting y fuerza a que la red no dependa de neuronas especificas. Ayuda a generalizar el modelo para datos nuevos.

Luego tengo otro bloque igual a las 3 capas anteriormente mencionadas solo que varia el fn.

2.3.5 Flatten - Output

Luego tengo una capa de flatten() que justamente se encarga de convertir un input (B,x,y) en algo (B,x×y).

Finalmente tengo una capa densa donde tiene tantas neuronas como clases de salidas. Usa una función de activación softmax que convierte lo que le llega a cada neurona en una probabilidad tal que la suma de $p_i = 1$.

2.4 Implementación en EI

En EI tenemos un par de parámetros para acomodar los cuales son:

- Epochs: Se refiere a cuantas veces se usan todos los datos para entrenar al modelo.
- Learned optimizer: Creo que usa un red neuronal como optimizador aunque esta parte no la entendí bien.
- Learning rate: "Que tan rápido aprende". Es un parámetro de que tanto se ajustan los pesos en cada iteración.
- Training processor: Es trivial.
- Batch size: La cantidad de batches que se usan en cada iteración antes de actualizar los pesos.
- Auto-weight classes: Nose como pero supuestamente "presta" mas atención a las clases menos importantes.

Ademas de eso tambien tenemos las opciones de **Data augmentation**:

- Mascara frecuencia
- Mascara temporal
- Ruido gaussiano
- Time warp

3 Clasificación

Una vez entrenado el impulso el procedimiento para clasificar es bastante similar.

- Se cargan audios de entrada de una determinada duración.
- Se calculan sus MFCC para cada ventana (misma que entrenamiento, segundo ventaneo)
- Se pasan esos datos al modelo (shape (B,50,13) si tomo 1er win=1s y 2win=20ms) B=length_audio/1s)
- Se obtiene para cada B un conjunto de prob (una por clase).
- Como la muestra es la union de todos los B, el resultado final depende de que la mayoria de los B pertenezcan a una clase :).