TALLER DE PROCESAMIENTO DE SEÑALES

1er Cuatrimestre 2026 - Trabajo Práctico Nº 3 - Regresión Logística

---

Se desea detectar comandos de voz de forma automática.

(**a**) *Preprocesamiento de datos:*

1. Descargar la base de datos `http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz`. Dicha descarga debe estar autocontenida en el código.
2. Utilizando `load` (librosa) cargar al menos tres audios de la base de datos para una primera exploración.
3. Reproducir los audios cargados previamente utilizando `Audio` (IPython).
4. Calcular la transformada de Fourier de corto tiempo de cada señal utilizando `stft` (librosa). Configurar dicha transformada para utilizar una FFT de 512 puntos, un solapamiento del 50 %, descartar la parte simétrica del espectro y utilizar ventana de *hann* de la misma cantidad de puntos de la fft. Explicar que significa cada parámetro mencionado.
5. Para cada ventana de tiempo, se desea descomponer el espectro de las señales en *bandas de energía*. Dividir el espectro en 20 bandas del mismo largo, y en cada una calcular la energía de la banda $i-$ésima $\mathcal{B}_i$ como:
$$E[i] = \sum_{k \in \mathcal{B}_i} |X[k]|^2$$
6. Convertir la energía en dB como:
$$E_{dB} = 10 \log_{10} \left(\frac{E}{E_{\text{máx}}} \right)$$
donde $E_{\text{máx}}$ es el máximo valor de energía de toda la señal.

Para el resto del tp solamente podrá importar os, numpy y matplotlib.pyplot (además de utilizar el código desarrollado en el inciso (a)).

(**b**) *Clasificación binaria:*

1. Construir una base de datos con los audios correspondientes a los comandos `YES` y `NO`.
2. Se desea unificar los largos de los audios, de manera que el procedimiento antes descripto genere 50 ventanas de tiempo. Recorte el final de los audios más largos y complete con ceros los más cortos.
3. Preprocesar los datos para quedarse con las bandas de energía en dB. : Cada muestra debe tener 50 ventanas de tiempo y 20 bandas de energía.
4. Definir los conjuntos de entrenamiento y testeo con las proporciones 80 % y 20 % de forma aleatoria.
5. Implementar una regresión logística binaria por gradiente descendente a partir de los datos generados previamente. El código debe estar estructurado de la siguiente manera:

```
class BinaryLogisticRegression:
    # Opcional, para inicializar atributos o declarar hiperparámetros
    def __init__(self,...

    # Etapa de entrenamiento
    def fit(self,X,y):

    # Etapa de testeo SOFT
    def predict_proba(self,X):

    # Etapa de testeo HARD
    def predict(self,X):

    # Cómputo del accuracy
    def accuracy(self,X,y):

    # Cómputo de la entropía cruzada
    def cross_entropy(self,X,y):
```

6. Graficar la entropía cruzada de entrenamiento en función de las iteraciones. Reajustar el *learning rate* y la cantidad de iteraciones hasta obtener (visualmente) cierta convergencia sin oscilaciones.

7. Reportar el *accuracy* y la entropía cruzada tanto para el conjunto de entrenamiento como el de testeo. Extraer conclusiones.

(**c**) *Clasificación categórica:*

1. Construir una base de datos con los audios correspondientes a los comandos `UP`, `DOWN`, `LEFT` y `RIGHT`.

2. Repetir los puntos $2-7$ del inciso (b) para regresión logística categórica.