

Procesamiento del Habla y Reconocimiento de Voz

OBS Business School

Introducción

- Automatic Speech Recognition
- Audio Classification
- Voice Activity Detection
- Diarización

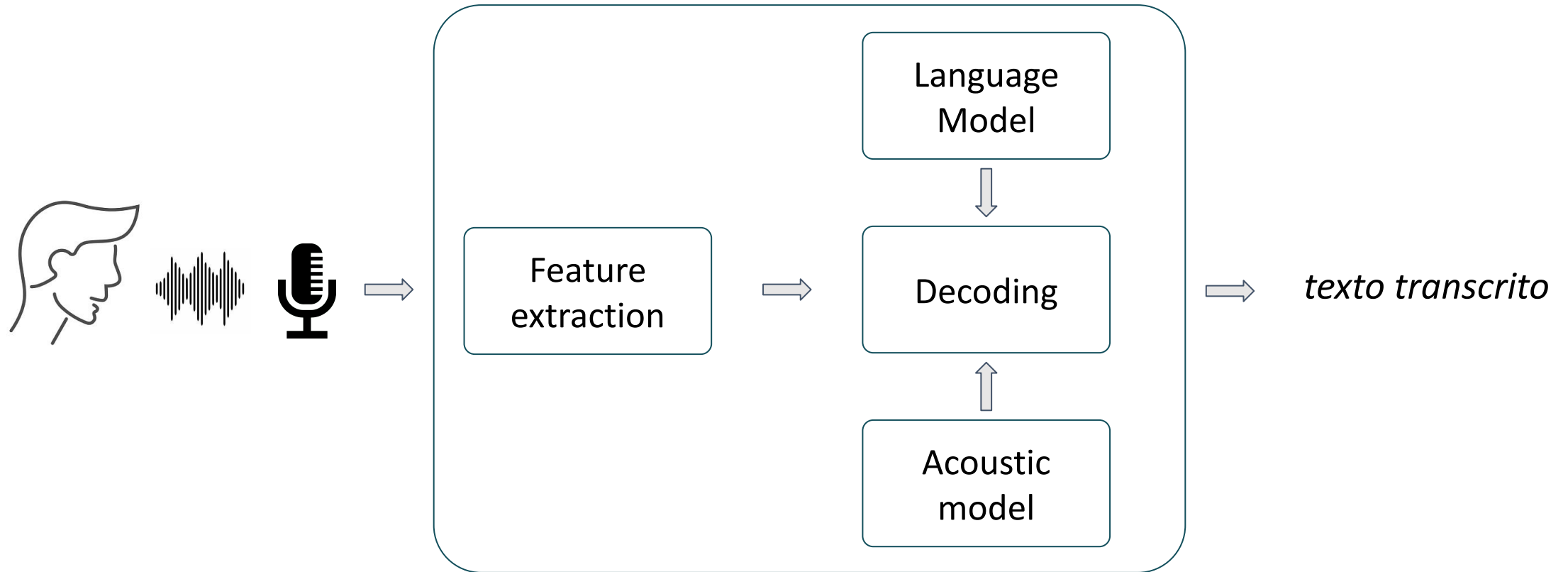
Automatic Speech Recognition

El Reconocimiento Automático del Habla (Automatic Speech Recognition, o ASR) es una tecnología que **transforma la voz humana en texto** utilizando algoritmos de procesamiento de señales y aprendizaje automático.



Debe tener en cuenta aspectos como:

- Acústica
 - Variabilidad entre hablantes (inter-speaker)
 - Variabilidad para el mismo hablante (intra-speaker)
 - Ruido, reverberación en la sala, entorno...
- Fonética
 - Articulación.
 - Elisiones (agrupar algunas palabras, no pronunciarlas).
 - Palabras con pronunciación similar.
- Lingüística
 - Amplitud del vocabulario.
 - Variaciones de palabras.
 - Palabras fuera del vocabulario.



La métrica más utilizada para medir el performance de un sistema de ASR está basada en la distancia de Levenshtein y es el **Word Error Rate (WER)**.

$$WER = \frac{N_{SUB} + N_{INS} + N_{DEL}}{|N_{words-transcript}|}$$

Donde,

- *Nwords-transcript*: número de palabras de la frase de referencia.
- *Nsub*: es el total de palabras que se sustituyen.
- *Nins*: es el total de palabras que no se pronunciaron pero se transcriben.
- *Ndel*: es el total de palabras que se omitieron en la transcripción.

Ejemplo:

- Frase real: “the cat sat on the mat”.
- Frase transcrita por un ASR: “the cat sit on the”.

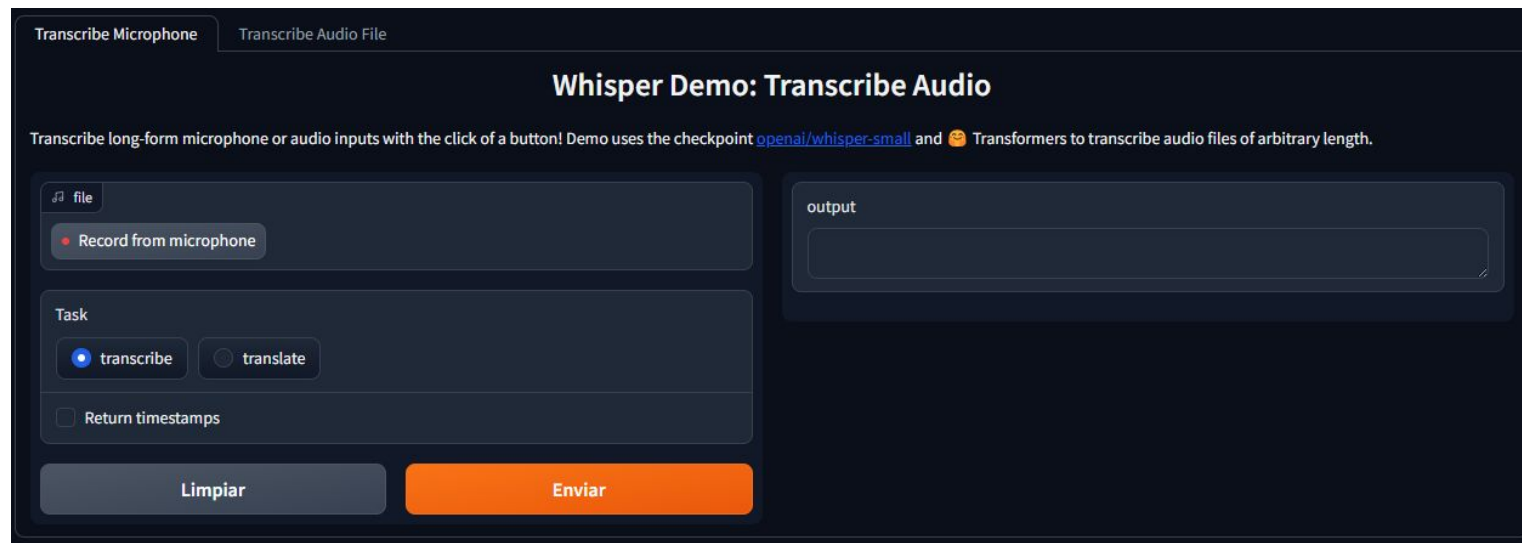
Reference:	the	cat	sat	on	the	mat
Prediction:	the	cat	sit	on	the	
Label:	✓	✓	S	✓	✓	D

$$\begin{aligned} WER &= \frac{S + I + D}{N} \\ &= \frac{1 + 0 + 1}{6} \\ &= 0.333 \end{aligned}$$

Link: <https://huggingface.co/learn/audio-course/chapter5/evaluation>

A continuación vamos a seguir su tutorial para **crear una aplicación web** que permita:

- **Grabar nuestra voz** con el micrófono.
- **Transcribir el audio a texto con Whisper** (OpenAI).
- **Visualizar los resultados.**



The screenshot shows a web application titled "Whisper Demo: Transcribe Audio". It has two tabs at the top: "Transcribe Microphone" (active) and "Transcribe Audio File". Below the tabs, there's a description: "Transcribe long-form microphone or audio inputs with the click of a button! Demo uses the checkpoint [openai/whisper-small](#) and 🧠 Transformers to transcribe audio files of arbitrary length." The interface includes a "file" input section with a "Record from microphone" button. Below that is a "Task" section with two radio buttons: "transcribe" (selected) and "translate". There is also a checkbox for "Return timestamps" which is currently unchecked. At the bottom, there are two buttons: "Limpiar" (grey) and "Enviar" (orange). On the right side, there is an "output" section with a large text area for displaying the transcription results.

Links:

- <https://huggingface.co/learn/audio-course/chapter5/demo>
- <https://course-demos-whisper-small.hf.space/>

1. Cargamos el modelo preentrenado usando Hugging Face.

```
from transformers import pipeline

model_id = "sanchit-gandhi/whisper-small-dv" # update with your model id
pipe = pipeline("automatic-speech-recognition", model=model_id)
```

2. Definimos una función que cargue un fichero de audio y lo transcriba a un idioma destino.

```
def transcribe_speech(filepath):
    output = pipe(
        filepath,
        max_new_tokens=256,
        generate_kwargs={
            "task": "transcribe",
            "language": "sinhalese",
        }, # update with the language you've fine-tuned on
        chunk_length_s=30,
        batch_size=8,
    )
    return output["text"]
```

3. Creamos una web app sencilla con Gradio para ejecutar el modelo.

```
import gradio as gr

demo = gr.Blocks()

mic_transcribe = gr.Interface(
    fn=transcribe_speech,
    inputs=gr.Audio(sources="microphone", type="filepath"),
    outputs=gr.outputs.Textbox(),
)

file_transcribe = gr.Interface(
    fn=transcribe_speech,
    inputs=gr.Audio(sources="upload", type="filepath"),
    outputs=gr.outputs.Textbox(),
)
```

Con Hugging Face Evaluate es posible calcular el WER de manera sencilla:

```
reference = "the cat sat on the mat"
```

```
prediction = "the cat sit on the"
```

```
pip install --upgrade evaluate jiwer
```

```
from evaluate import load

wer_metric = load("wer")

wer = wer_metric.compute(references=[reference], predictions=[prediction])

print(wer)
```

```
0.3333333333333333
```

Link: <https://huggingface.co/learn/audio-course/chapter5/evaluation>

4. Lanzamos la aplicación de Gradio.

```
with demo:  
    gr.TabbedInterface(  
        [mic_transcribe, file_transcribe],  
        ["Transcribe Microphone", "Transcribe Audio File"],  
    )  
  
demo.launch(debug=True)
```

The screenshot shows a web application titled "Whisper Demo: Transcribe Audio". It has two tabs at the top: "Transcribe Microphone" (active) and "Transcribe Audio File". The main text says: "Transcribe long-form microphone or audio inputs with the click of a button! Demo uses the checkpoint [openai/whisper-small](#) and 🧠 Transformers to transcribe audio files of arbitrary length."

Under the "Transcribe Microphone" tab, there is a section with a "file" icon and a "Record from microphone" button. Below this is a "Task" section with two radio buttons: "transcribe" (selected) and "translate". There is also a checkbox for "Return timestamps" which is currently unchecked. At the bottom of this section are two buttons: "Limpiar" (grey) and "Enviar" (orange).

On the right side, there is an "output" section with a large text area for the transcription results.

Audio Classification

La clasificación de audio es un proceso que consiste en **identificar y categorizar segmentos de audio según su contenido sonoro**.

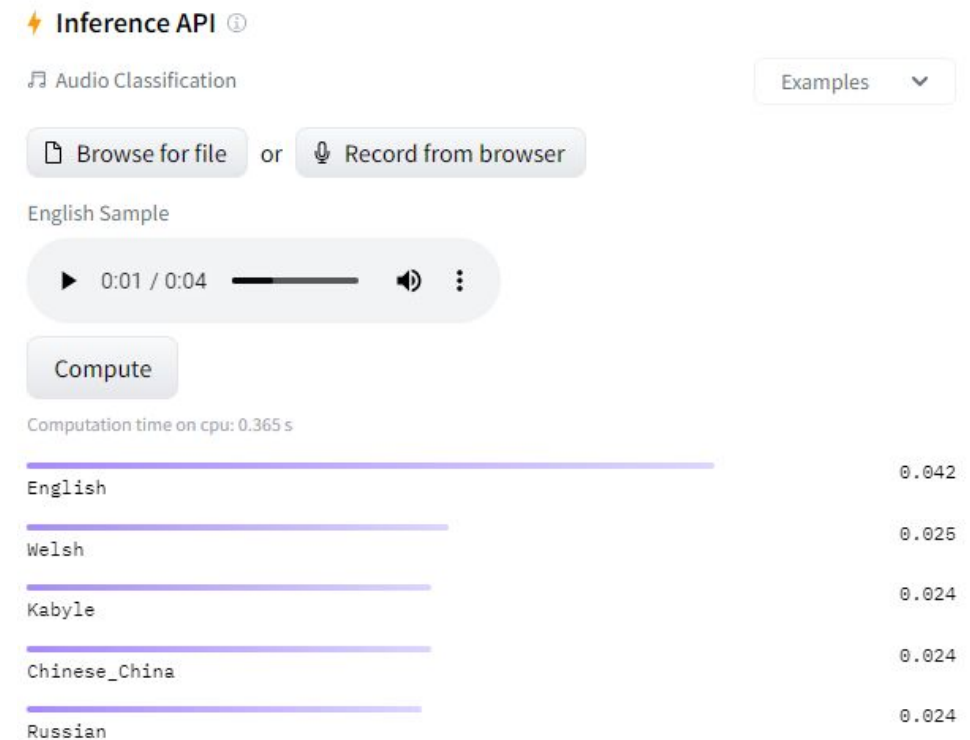
Tiene múltiples aplicaciones:

- **Reconocimiento de voz:** Identifica la presencia y origen de voces en segmentos de audio.
- **Detección de eventos sonoros:** Reconocimiento de determinados sonidos en entornos ruidosos, para seguridad y monitoreo.
- **Análisis de sentimiento:** Identificación del tono emocional y los sentimientos expresados en grabaciones de voz.
- **Clasificación de música:** Diferencia géneros musicales y características como instrumentos o ritmo.
- **Diagnóstico por sonido:** En industria y medicina para el diagnóstico de máquinas o condiciones de salud a través de análisis de sonidos.
- **Mejora de la calidad del audio:** Filtrando ruidos no deseados.
- **Bioacústica:** Monitoreo, estudio y conservación de especies silvestres identificando sus sonidos.

- La clasificación del audio puede realizarse para detectar automáticamente el idioma empleado.
- Por ejemplo, SpeechBrain ofrece en Hugging Face [1] ofrece un modelo entrenado a partir del dataset Common Language [2] que permite clasificar audios en 45 idiomas.

[1] https://huggingface.co/speechbrain/lang-id-commonlanguage_ecapa

[2] https://huggingface.co/datasets/speechbrain/common_language



- Common Voice es un proyecto de Mozilla que busca crear una **base de datos de grabaciones de voz de libre uso y acceso público** para ayudar en el desarrollo y la investigación de tecnologías de reconocimiento de voz.
- Recopila **voces de voluntarios de todo el mundo** contribuyendo a la **diversidad** asegurando que la efectividad de estos sistemas para personas de diferentes géneros, acentos y lenguas.
- Es posible **donar grabaciones de voz** o ayudando a **verificar las grabaciones de otros**.
- Link: <https://commonvoice.mozilla.org/es>



- El modelo Audio Spectrogram Transformer (AST) [1] permite clasificar audios en multitud de etiquetas correspondientes a distintos tipos de sonidos.
- Ha sido entrenado con el dataset AudioSet.

[1] <https://huggingface.co/MIT/ast-finetuned-audioset-10-10-0.4593>

⚡ Inference API ⓘ

🎵 Audio Classification

📁 Browse for file or 🎙️ Record from browser

Audio recorded from browser [3:51:40 PM]

▶ 0:04 / 0:04 🔊 ⋮

Compute

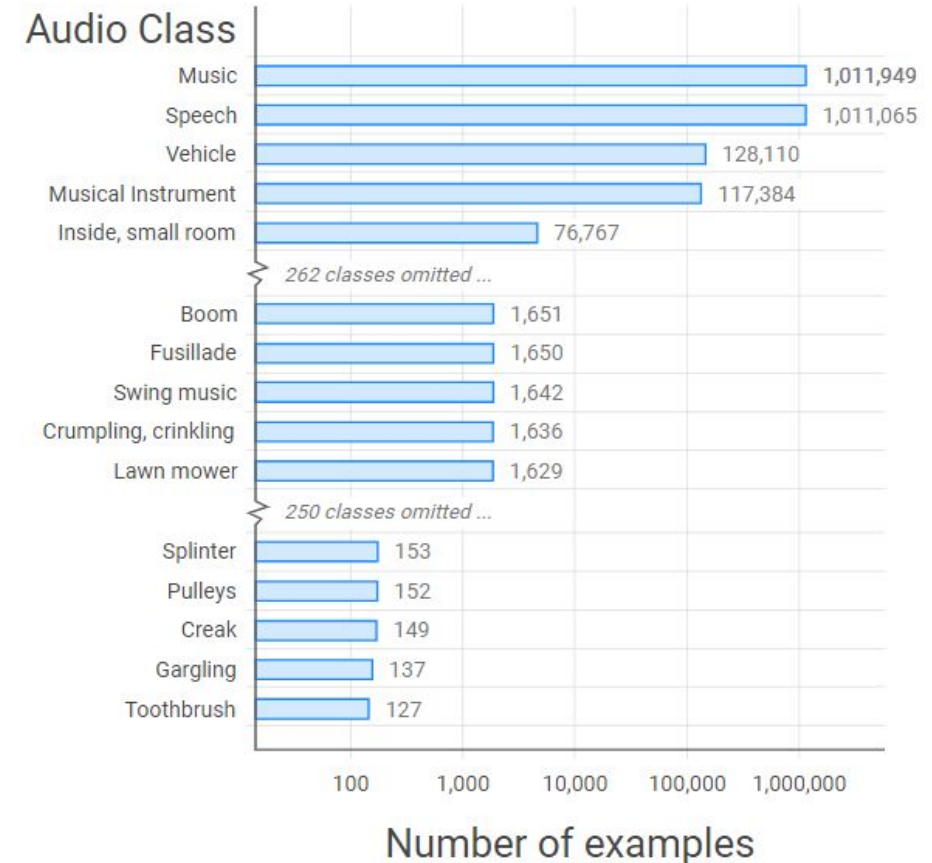
Computation time on cpu: 1.677 s

Speech	0.913
Speech synthesizer	0.017
Clicking	0.010
Male speech, man speaking	0.006
Tick	0.004

🔗 JSON Output

🔍 Maximize

- VoiceSet es un proyecto de Google Voice que no se centra exclusivamente en la voz humana, sino que **recopila una amplia variedad de sonidos del entorno.**
- Incluye **más de 2 millones de clips de sonido** humanamente anotados que abarcan **632 categorías de eventos sonoros** como música, ruido ambiental, sonidos de animales, y muchos otros.
- De manera similar a Common Voice, anima a voluntarios de todo el mundo a contribuir con sus grabaciones de voz.
- Link: <https://research.google.com/audioset/>

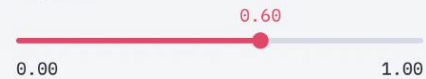


Voice Activity Detection

- Voice Activity Detection se refiere a las tecnologías que se emplean para determinar si en un segmento de audio está presente la voz humana o no.
- Se trata de una etapa fundamental en contextos donde es necesario diferenciar entre el habla y el silencio o ruido de fondo.
- También, es ampliamente utilizado en telecomunicaciones, reconocimiento de voz, y en aplicaciones que requieren la compresión de datos de audio.

▶ 0:02 / 0:02 — 🔊 ⋮

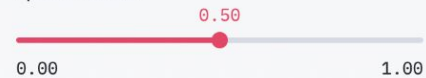
Threshold



Window size



Speech Window



Based on :

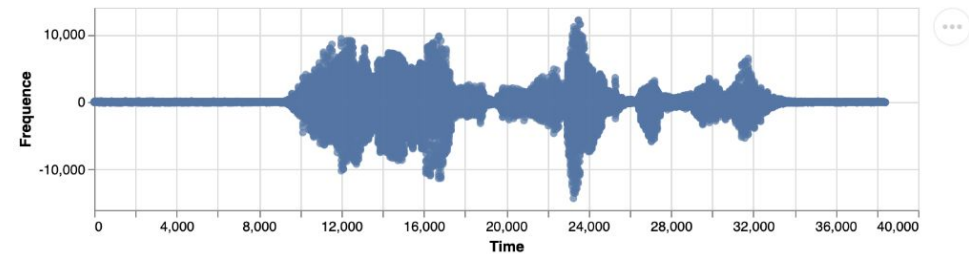
<https://github.com/marsbroshok/VAD-python>

Input audio data treated as following:

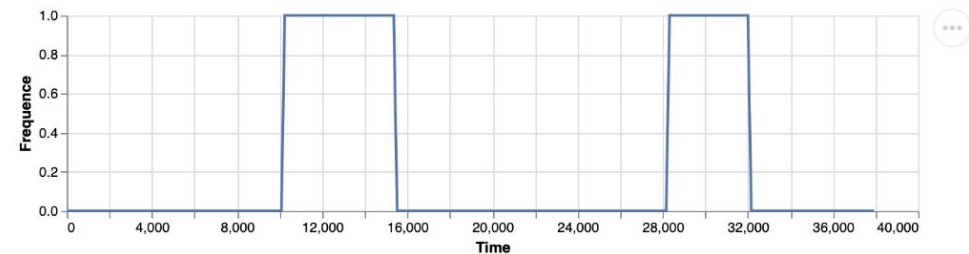
- Convert stereo to mono

This application demonstrates a simple Voice Activity Detection algorithm that works for any language.

Plot the raw audio signal:

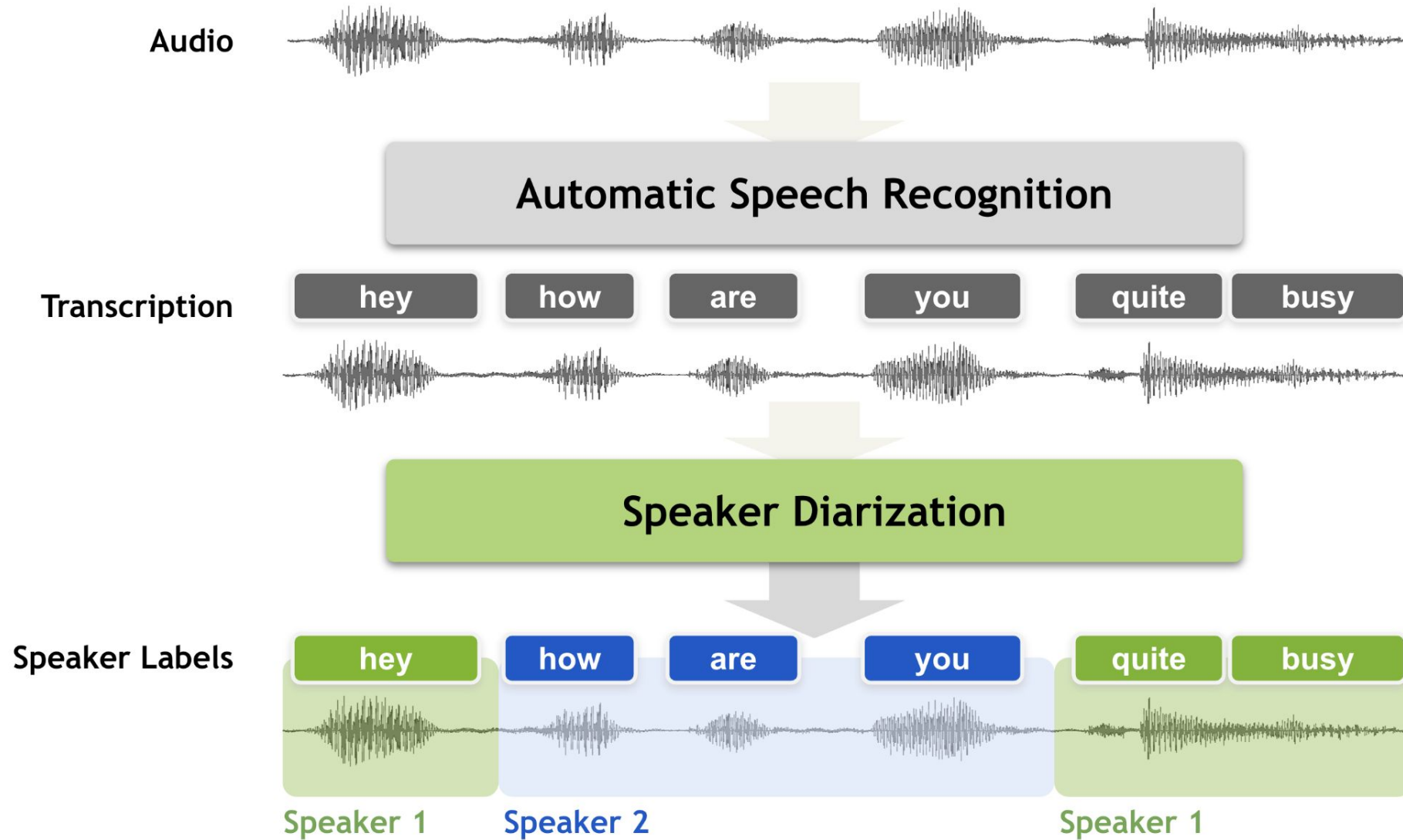


Plot the detected voice:



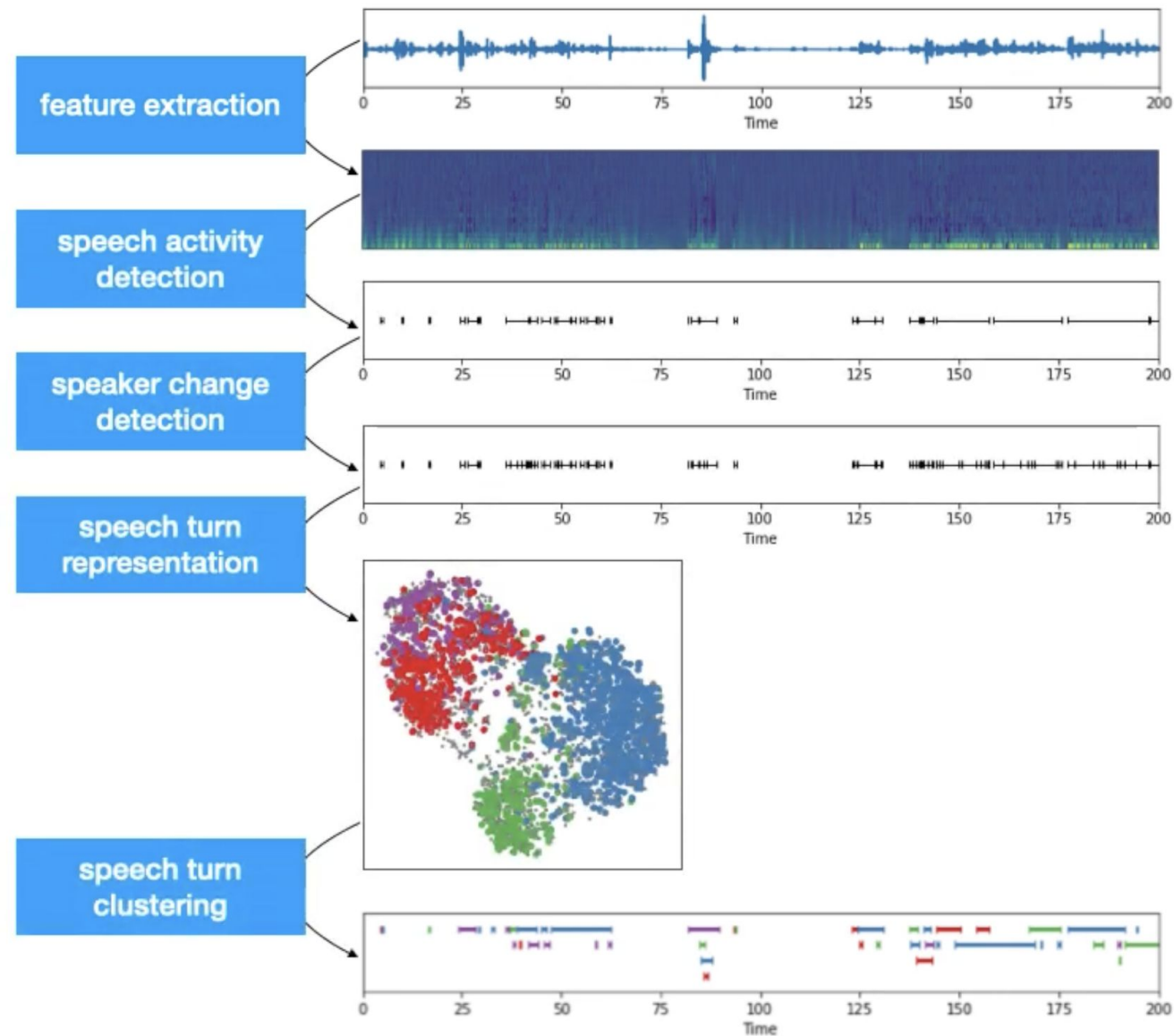
Link: <https://maelfabien.github.io/machinelearning/Speech4/#high-level-overview>

- La diarización es el proceso utilizado en el reconocimiento del habla para determinar "quién habla cuándo" en una grabación que contiene múltiples hablantes.
- Este proceso divide un flujo de audio en secciones que corresponden a diferentes.
- La diarización es útil en aplicaciones como la transcripción automatizada de reuniones, entrevistas, llamadas de servicio al cliente y programas de radio o televisión donde intervienen varios participantes.

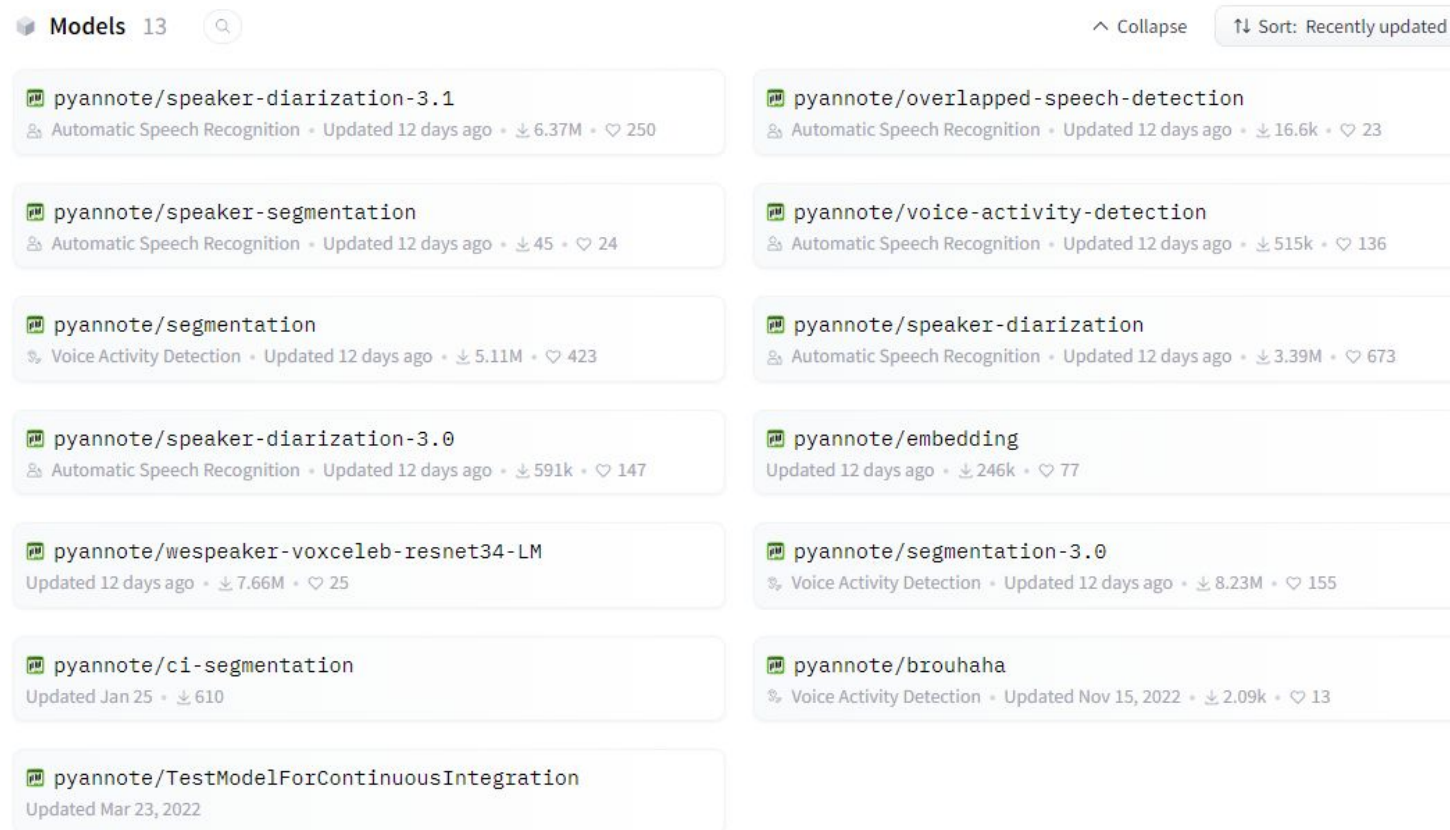


El proceso de diarización se compone de los siguientes pasos:

1. **Detección de actividad vocal:** Se identifica cuándo hay habla y cuándo no.
2. **Segmentación del habla:** El audio con habla detectada se divide en segmentos más pequeños que se presume contienen habla de un solo hablante.
3. **Agrupación de segmentos:** Los segmentos se comparan y se agrupan según características similares, presumiblemente correspondientes al mismo hablante.



- pyannote.audio es un framework para la diarización de audios en Python.
- Tiene disponible en Hugging Face distintos modelos preentrenados.



Links: <https://huggingface.co/pyannote>, <https://github.com/pyannote/pyannote-audio>

- Es posible representar a un interlocutor mediante su embedding.
- pyannote.audio ofrece un modelo para tal finl.

```
# 1. visit hf.co/pyannote/embedding and accept user conditions  
# 2. visit hf.co/settings/tokens to create an access token  
# 3. instantiate pretrained model  
from pyannote.audio import Model  
model = Model.from_pretrained("pyannote/embedding",  
                             use_auth_token="ACCESS_TOKEN_GOES_HERE")
```

```
from pyannote.audio import Inference  
inference = Inference(model, window="whole")  
embedding1 = inference("speaker1.wav")  
embedding2 = inference("speaker2.wav")  
# 'embeddingX' is (1 x D) numpy array extracted from the file as a whole.  
  
from scipy.spatial.distance import cdist  
distance = cdist(embedding1, embedding2, metric="cosine")[0,0]  
# 'distance' is a 'float' describing how dissimilar speakers 1 and 2 are.
```

Links: <https://huggingface.co/pyannote>, <https://github.com/pyannote/pyannote-audio>

- pyannote.audio ofrece un modelo preentrenado para la diarización automática de audios.

1. Install `pyannote.audio` with `pip install pyannote.audio`
2. Accept `pyannote/segmentation-3.0` user conditions
3. Accept `pyannote/speaker-diarization-3.1` user conditions
4. Create access token at `hf.co/settings/tokens`.

```
from pyannote.audio import Pipeline
pipeline = Pipeline.from_pretrained(
    "pyannote/speaker-diarization-3.1",
    use_auth_token="HUGGINGFACE_ACCESS_TOKEN_GOES_HERE")

# send pipeline to GPU (when available)
import torch
pipeline.to(torch.device("cuda"))

# apply pretrained pipeline
diarization = pipeline("audio.wav")

# print the result
for turn, _, speaker in diarization.itertracks(yield_label=True):
    print(f"start={turn.start:.1f}s stop={turn.end:.1f}s speaker_{speaker}")
# start=0.2s stop=1.5s speaker_0
# start=1.8s stop=3.9s speaker_1
# start=4.2s stop=5.7s speaker_0
# ...
```

Links: <https://huggingface.co/pyannote>, <https://github.com/pyannote/pyannote-audio>

Próximos pasos



Forbes



Ranking Educativo
Innovatec



EL MUNDO



- Repaso y lectura de los conceptos tratados en la 2ª sesión.
- Próxima sesión: miércoles 29 de mayo a las 19:00 (CEST).
- Dudas y preguntas, vía *tablero de discusión* o email.

OBS Business
School



Planeta Formación y Universidades